Otimização de Processos de Picking

Licenciatura em Engenharia Informática

André Miguel Romeiro Faria Lopes

Nuno Miguel Hilário Caseiro

Leiria, julho de 2020

Otimização de Processos de Picking

Licenciatura em Engenharia Informática

André Miguel Romeiro Faria Lopes

Nuno Miguel Hilário Caseiro

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação do Professor Carlos Fernando de Almeida Grilo, do Professor Luís Miguel Pires Neves e da Professora Eunice Sandra Gomes de Oliveira.

Leiria, julho de 2020

Agradecimentos

Gostaríamos de expressar os nossos sinceros agradecimentos aos orientadores Professor Carlos Fernando de Almeida Grilo, Professor Luís Miguel Pires Neves e Professora Eunice Sandra Gomes de Oliveira pela indispensável ajuda, partilha de conhecimento e disponibilidade mostrada ao longo do desenvolvimento deste projeto.

Resumo

Este projeto, aborda o tema da otimização de processos de *picking*, que consiste na recolha de produtos das prateleiras de armazéns e que é uma tarefa essencial e dispendiosa nas operações diárias de um armazém, sendo estimado que representa 55% dos custos operacionais. Neste projeto, desenvolveu-se uma aplicação que procura responder às necessidades de armazéns, através da distribuição de produtos por diferentes operadores, tendo em conta possíveis colisões e as restrições de peso dos produtos e dos agentes, tal como a ordem pela qual estes produtos são recolhidos, resultando assim, numa diminuição do tempo de recolha dos mesmos. Esta aplicação, permite ainda que os utilizadores testem diferentes combinações de parâmetros do algoritmo desenvolvido e visualizem a solução obtida, bem como a realização de experiências automatizadas. Durante o processo de desenvolvimento, utilizaram-se conceitos das diversas metodologias ágeis de desenvolvimento de *software* nomeadamente *Scrum, Kanban e Extreme programming*. Concluiu-se com este projeto que o tempo de recolha dos produtos diminui com o número de agentes e aumenta com o número de produtos de forma aproximadamente linear. Para além disso, o algoritmo desenvolvido permite obter soluções com um baixo número de colisões.

Palavras-chave: *Picking*, *Warehouse*, Algoritmos genéticos, A*, Colisões, Restrições

Abstract

This project addresses the optimization of picking processes which consist in the collection of products from shelves and is an essential and costly task in the daily operations of a warehouse, representing around 55% of the operational costs. The purpose of this project is to develop an application that matches the necessities of warehouses by assigning products to different agents, taking into account collisions, product weight restrictions, agent capacity, as well as the order by which products are picked, resulting in the reduction of the time needed to perform this task. This application also allows users to test different parameter combinations for the algorithm developed and visualize the resulting solution as well as perform experiments. In the development of this project, were used concepts from several agile software development methodologies namely Scrum, Kanban, and Extreme programming. The main conclusions from this project are that the time to collect the products decreases with a higher number of agents and increases approximately linearly when the number of agents also increases. Furthermore, the algorithm developed generates solutions with a low number of collisions.

Keywords: Picking, Warehouse, Genetic Algorithms, A*, Collisions, Restrictions

Lista de Figuras

Figura 1 - Pseudocódigo do algoritmo genético 7

Figura 2 - Pseudocódigo algoritmo de procura 9

Figura 3 - Representação do quadro Kanban 11

Figura 4 - Diagrama de implementação 14

Figura 5 - Representação do armazém 16

Figura 6 - Representação do ficheiro do armazém (esquerda) e do ficheiro dos *picks* (direita) 17

Figura 7 - Representação de um indivíduo 19

Figura 8 - Representação de uma colisão entre dois agentes 21

Figura 9 - Tempo máximo para resolver uma colisão usando a segunda abordagem (esquerda) e usando a terceira abordagem (direita) 22

Figura 10 – Método de seleção por torneio 24

Figura 11 - Seleção baseada em ordenação 25

Figura 12 - Representação do método de recombinação PMX. 27

Figura 13 - Representação do operador OX 28

Figura 14 - Representação do operador *OX*1 29

Figura 15 - Representação do operador CX 30

Figura 16 - Representação do operador *Insert* 31

Figura 17 - Representação do operador *Inversion* 31

Figura 18 - Representação do operador S*cramble* 32

Figura 19 - Representação do padrão *Observer* 33

Figura 20 - Representação da interface gráfica e vista do algoritmo genético 35

Figura 21 - Representação da vista da simulação 36

Figura 22 - Representação da vista das experiências 37

Figura 23 - Representação da janela de erros 37

Figura 24 - Representação do armazém 38

Figura 25 - Representação do tempo e das colisões para diferentes ponderações 40

Figura 26 - Representação do tempo em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para as combinações de ponderações (8, 6) (esquerda) e (4, 10) (direita) 41

Figura 27 - Representação do tempo em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para as combinações de ponderações (8, 6) (esquerda) e (4, 10) (direita) 41

Figura 28 - Representação do número de colisões em função do número de *picks* e agentes tendo em conta a capacidade dos agentes e produtos para as ponderações (8, 6) (esquerda) e (4, 10) (direita) 42

Figura 29 - Representação do número de colisões em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para as ponderações (8, 6) (esquerda) e (4, 10) (direita) 42

Figura 30 - Representação do número de deslocações ao ponto de entrega por cada agente em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para as ponderações (8, 6) (esquerda) e (4, 10) (direita) 43

Figura 31 - Representação do número de deslocações ao ponto de entrega por cada agente em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para as ponderações (8, 6) (esquerda) e (4, 10) (direita) 44

Figura 32 - Representação do tempo em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens 44

Figura 33 - Representação do tempo em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para a segunda (esquerda) e terceira (direita) abordagens 45

Figura 34 - Representação do número de colisões em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens 45

Figura 35 - Representação do número de colisões em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para a segunda (esquerda) e terceira (direita) abordagens 46

Figura 36 - Representação do número de deslocações ao ponto de entrega por cada agente em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens 46

Figura 37 - Representação do número de deslocações ao ponto de entrega por cada agente em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens 47

Figura 38 - Representação da média da soma dos tempos de espera em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens 47

Figura 39 - Representação da média da soma dos tempos de espera em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para a segunda (esquerda) e terceira (direita) abordagens 48

Figura 40 - Representação do tempo de espera máximo em função do número de *picks* e agentes em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens 48

Figura 41 - Representação do tempo de espera máximo em função do número de *picks* e agentes em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para a segunda (esquerda) e terceira (direita) abordagens 49

Lista de siglas e acrónimos

ADN	Ácido desoxirribonucleico
CX	*Cycle Crossover*
ESTG	Escola Superior de Tecnologia e Gestão
GUI	*Graphical User Interface*
IPLeiria	Instituto Politécnico de Leiria
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
MVC	*Model View Controller*
OPS	*Order Picking Systems*
OX	*Order Crossover*
PMX	*Partially Mapped Crossover*
TSP	*Traveling Salesman Problem*

Índice

1. Introdução 1

2. Estado da Arte 3

3. Teoria de suporte 5

3.1. Algoritmos genéticos 5

3.1.1. Funcionamento do algoritmo genético 6

3.2. Algoritmos de procura exaustiva 7

4. Metodologias de desenvolvimento 10

5. Arquitetura de software 12

5.1. Requisitos de alto nível 12

5.2. Diagrama de implementação 13

5.3. Tecnologias utilizadas 14

6. Solução desenvolvida 15

6.1. Problema a resolver 15

6.2. Representação do armazém 15

6.3. Representação do armazém em ficheiro 17

6.4. A* 18

6.5. Algoritmo genético 18

6.5.1. Representação dos indivíduos 19

6.5.2. Função de avaliação 19

6.5.3. Operadores de seleção 23

6.5.4. Operadores de recombinação 26

6.5.5. Operadores de mutação 30

6.6. Módulo para realização de experiências 32

6.7. Independência do modelo 33

6.8. Testes de software 34

6.9. Descrição da interface gráfica de utilizador 34

7. Experiências e resultados 38

7.1. Abordagem baseada em ponderações 41

7.2. Abordagens que penalizam o tempo de espera para evitar colisões 44

7.3. Discussão de resultados 49

8. Conclusão 51

Apêndice A – Testes manuais realizados 54

Apêndice B – Manual de utilizador 55

1. Introdução

O processo de *picking* num armazém consiste na atribuição de um conjunto de produtos a recolher, aos diferentes operadores, bem como a definição da ordem pela qual cada operador deve recolher os produtos.

O processo de *picking* pode parecer simples, no entanto, estima-se que representa em média 55% dos custos operacionais em qualquer centro de distribuição, sendo um aspeto essencial no atendimento de pedidos e é considerada uma das atividades mais caras e intensas para os armazéns. Assim, é essencial desenvolver uma estratégia que permita aumentar a rapidez, precisão e organização do processo, estratégia essa que pode fazer a diferença relativamente a empresas concorrentes.

Este relatório explora o tema da otimização de processos de *picking* e foi desenvolvido no âmbito da unidade curricular de projeto informático, no segundo semestre do terceiro ano da licenciatura em Engenharia Informática na Escola Superior de Tecnologia e Gestão (ESTG) do Politécnico de Leiria no ano letivo de 2019/2020.

A otimização deste processo resulta numa diminuição do tempo de recolha dos produtos de uma certa encomenda num armazém, por uma ou mais pessoas, tendo em conta algumas características dos objetos com prioridades bem definidas. Esta diminuição do tempo de recolha, por sua vez, leva à possibilidade de se ter menos funcionários a desempenhar a tarefa, o que resulta na poupança de dinheiro. Este projeto pode também ser aplicado a armazéns automatizados, isto é, com operadores não humanos.

A recolha dos itens afeta diretamente as operações de um armazém e pode afetar outras áreas de negócio. A velocidade e desempenho dos operadores dos pedidos são aspetos a ter em conta uma vez estes têm um enorme papel para manter um bom nível operacional dentro do armazém e influenciar os níveis de satisfação do cliente. Contratempos no envio e receção da encomenda pode levar o cliente a procurar alternativas.

Como um operador de armazém passa a maior parte do tempo a deslocar-se, uma das maiores causas de ineficiência do processo é o desperdício de movimento, ou seja, deslocações desnecessárias que levam a um aumento de tempo na execução da tarefa. Porém, alguns

armazéns possuem tecnologia capaz de ajudar os trabalhadores a otimizar o seu caminho de forma a minimizar a distância e tempo necessário, bem como como evitar *bottlenecks* e colisões.

Neste projeto mostramos como foi implementada uma variação do conhecido problema *travelling salesman* aplicado à otimização do processo de *picking* em armazéns recorrendo a técnicas heurísticas, otimização esta que consiste em encontrar uma forma de minimizar o tempo despendido na recolha de itens no processo de *picking*, procurando evitar colisões entre os operadores, bem como lidar com as restrições de peso e capacidade de cada produto.

É de realçar, que o propósito futuro deste projeto, é ser integrado com outros sistemas e funcionar sem a necessidade da interação humana, pelo que, a interface gráfica desenvolvida serve exclusivamente como protótipo para testar o algoritmo desenvolvido.

Os restantes capítulos do relatório estão estruturados da seguinte forma: No Capítulo 2, é apresentada uma visão geral do problema, outras propostas de resolução do problema e as escolhas adotadas com base em referências bibliográficas; O Capítulo 3, tem como propósito expor os conceitos necessários para uma melhor compreensão dos algoritmos utilizados; No Capítulo 4, são descritas as metodologias de desenvolvimento de software utilizadas; O Capítulo 5, destaca as escolhas a nível de padrões de software e tecnologias utilizadas; No Capítulo 6 são descritas e justificadas as diferentes abordagens adotadas no desenvolvimento do projeto; No Capítulo 7 são apresentadas as experiências efetuadas e os resultados obtidos; Finalmente, no Capítulo 8 são expostas as conclusões retiradas da elaboração do projeto e atividades futuras.

2. Estado da Arte

O problema de *picking*, é um problema bem estudado e consiste na recolha de produtos que se encontram em prateleiras de um armazém por um conjunto de agentes (operadores que recolhem produtos das prateleiras). O processo de *picking* pode ser realizado consoante várias abordagens de alto nível, designadas por *order picking systems* (OPS). Dallari et al. [1] propõem a classificação de OPS em quatro grupos: *Picker-to-parts*, que representam a grande maioria de sistemas em armazéns, onde os *pickers* se deslocam pelos corredores para recolherem itens de uma ou mais encomendas; *Pick-to-box*, onde a área de recolha é dividida em zonas, que são atribuídas a um ou mais *pickers*, todas estas zonas estão ligadas via um transportador (e.g. tapete rolante) onde são colocadas as caixas com os produtos recolhidos em cada área; *Pick-and-sort*, em que os operadores recolhem a quantidade de um único item, resultante de um conjunto de encomendas (i.e. pelo menos 20), e os colocam num transportador, que liga a área de recolha do armazém à área de triagem. Este transportador contém mecanismos que direcionam cada item, para um grupo de produtos pertencentes à mesma ordem; E, por fim, *parts-to-picker* onde dispositivos automáticos, trazem itens até às estações de *picking* onde os *pickers* selecionam a quantidade necessária de cada produto para uma certa encomenda.

Para o problema de *picking* existem várias variantes que são abordadas em diversos estudos, sendo que este projeto aborda uma variante que pode ser classificada como *picker-to-parts*. Este problema pode ser resolvido através de um único agente, tendo como inspiração a resolução do problema *Traveling Salesman Problem* (TSP) como é explorado em [2] ou com múltiplos agentes tentando obter a solução ótima recorrendo a algoritmos de procura como é demonstrado em [3]. Em problemas que utilizam uma abordagem multiagente, pode ser necessário considerar as colisões entre os agentes. Os problemas em que as colisões são consideradas, merecem especial atenção, sendo que vários artigos exploram e apresentam soluções [3,4].

No artigo [3] , as colisões são classificadas em dois tipos: colisões em nós, quando dois agentes se intersetam no mesmo nó; colisões em caminho, quando dois agentes se encontram no mesmo caminho, ao mesmo tempo e em direções opostas acabando por se encontrarem em nós diferentes. Com base nos tipos de conflitos que o autor do artigo descreve, são propostas diferentes abordagens para lidar com as colisões. A primeira solução consiste na espera de um dos agentes envolvidos na colisão até que o outro liberte o caminho, sendo atribuído um tempo

de espera diferente para cada tipo de colisão. O tempo de espera é atribuído consoante o tempo que o outro agente permanece no caminho ou um tempo de espera baixo se a colisão acontecer num nó. A segunda solução admite a alteração do caminho de um agente, utilizando a versão melhorada do algoritmo *Dijkstra*. Para um agente é calculado um novo caminho mais curto até ao nó objetivo e caso a colisão não tenha sido resolvida a rota do agente é recalculada com base numa nova representação do ambiente. Isto é, para os conflitos em nós, o nó onde decorre a colisão e os seus sucessores são removidos. Para os conflitos em caminho, a aresta é removida. A última solução passa por recalcular todo o trajeto para um determinado agente. O autor sugere a utilização em conjunto da primeira e segunda solução.

No problema de *picking*, outra variante é a consideração dos pesos e capacidades máximas que os produtos suportam, para não se danificarem (fragilidade) e o limite de peso dos monta cargas (dirigido pelo agente). No artigo [5] é descrita uma solução para esta variante do problema. Os produtos são classificados como frágeis ou pesados. Assim, os produtos pesados não devem ser colocados em cima dos frágeis. Quando excedida a capacidade dos monta cargas apenas poderão ser colocados produtos leves.

No que diz respeito aos algoritmos usados para resolver este problema, podem ser encontradas várias abordagens na literatura, sendo algumas destas: algoritmos exatos [6] que geram soluções ótimas. Heurísticas [7, 8] específicas para cada problema que não garantem a obtenção de soluções ótimas. E meta-heurísticas [9, 10], que também não garantem soluções ótimas, mas cuja conceção genérica é adaptável a diferentes problemas. Em [11] conclui-se que, consoante o tamanho do problema, os algoritmos exatos podem ser incapazes de encontrar uma solução em tempo útil. Em [12] são comparadas heurísticas e meta-heurísticas, aplicadas a problemas de otimização e é demostrado que as meta-heurísticas como, por exemplo, algoritmos genéticos, produzem habitualmente as melhores soluções.

3. Teoria de suporte

Este capítulo, tal como o título indica, tem como propósito expor os conceitos necessários para uma melhor compreensão dos algoritmos utilizados no projeto. Para o efeito, o capítulo é dividido em duas secções: na primeira, faz-se uma breve introdução aos algoritmos genéticos, inspirados na teoria da evolução das espécies de Charles Darwin, sendo abordado o funcionamento de um algoritmo genético e descritas algumas terminologias, definidas em analogia com os seres vivos; na segunda secção, pretende-se demonstrar como os algoritmos de procura surgem na resolução de problemas e o seu funcionamento.

3.1. Algoritmos genéticos

Em 1859 Charles Darwin publicou o livro *On the Origin of Species*, que expôs a sua teoria concebida maioritariamente após observações, durante o período em que este fez parte de uma expedição que incluiu paragens na América do Sul, Austrália, e Sul de África. Esta teoria propõe que as espécies evoluem de tal maneira que os seres vivos se tornam mais adaptados ao ambiente onde vivem e que esta evolução resulta do mecanismo de seleção natural.

Darwin propôs que os seres vivos evoluem através de seleção natural. Apesar de não saber o motivo, Darwin observou que nos seres vivos muitas características são herdadas e que certas características fazem com que os indivíduos sobrevivam mais facilmente e se reproduzam com mais frequência. Como os indivíduos são capazes de produzir mais descendentes do que o ambiente onde se encontram consegue suportar, vai existir competição pelos recursos limitados existentes, fazendo com que apenas os indivíduos mais aptos ao ambiente sobrevivam e por sua vez se reproduzam mais. Logo, ao longo do tempo as características destes tornam-se mais prevalentes na população tornando assim a população mais adaptada ao ambiente onde vive.

A teoria moderna de evolução suporta a seleção natural complementando-a com os conhecimentos da área da genética. Esta propõe que a evolução é a alteração da frequência de certos alelos numa certa população ao longo do tempo. Um indivíduo apenas possui uma porção dos alelos da população que são herdados dos pais e sujeitos a mutações e recombinações introduzindo assim diversidade na população. A seleção natural favorece certos indivíduos que por sua vez vão contribuir com uma maior porção dos seus genes para a próxima geração.

Os algoritmos genéticos são meta-heurísticas (técnicas de otimização estocásticas), propostas por John Holland e inspiradas no processo de evolução através de seleção natural, tal como

sugerida por Charles Darwin. Estes são normalmente usados para gerar soluções para problemas de otimização e procura, usando operadores inspirados na biologia como a seleção, mutação e recombinação.

Inspirados nos processos biológicos da genética das populações, os algoritmos genéticos utilizam uma terminologia que apresentamos a seguir.

Todos os seres vivos são constituídos por células que contêm um conjunto de cromossomas. Estes representam cadeias de ácido desoxirribonucleico (ADN) que são constituídas por genes. Os genes codificam características dos indivíduos. Aos diferentes valores que cada gene pode tomar dá-se o nome de alelos. O conjunto particular de genes contidos no genoma designa-se por genótipo. Nos algoritmos genéticos estes genes codificam também características da solução, sendo representados através de uma sequência de valores onde cada um destes corresponde a uma das variáveis de decisão.

Um algoritmo genético, trabalha sobre uma população de indivíduos, cujo tamanho permanece geralmente constante ao longo das gerações. Cada indivíduo representa uma solução candidata para um problema.

A qualidade de um indivíduo é representada pelo resultado de uma função de avaliação, que quantifica a capacidade que este tem para produzir descendência e eventualmente permanecer na geração seguinte. Os melhores indivíduos, terão mais probabilidade de se reproduzir e assim afirmar as suas características na população.

O mecanismo de recombinação, consiste na permuta de material genético entre dois progenitores, selecionados de acordo com a função de avaliação. Os novos indivíduos, podem ser sujeitos a mutações, que provocam troca dos valores de um ou mais genes.

As implementações dos algoritmos genéticos introduzem processos que não seguem de forma precisa os fenómenos naturais, mas que se esperam acelerar o processo de procura de soluções, evitando percorrer partes do espaço de soluções com menor probabilidade de conterem solução ótima.

3.1.1. Funcionamento do algoritmo genético

O algoritmo genético clássico como referido em [13], funciona da seguinte forma: a população inicial é gerada aleatoriamente, com um certo número de indivíduos. A população inicial evolui ao longo de um número de gerações através da aplicação de operadores de seleção,

recombinação e mutação. Os indivíduos com o melhor *fitness* (que apresentam a melhor avaliação da solução ao problema que descrevem), têm uma maior probabilidade de criar descendentes através de operadores de recombinação. Para a escolha dos indivíduos, que vão produzir a próxima geração, recorre-se a operadores de seleção. Através da aplicação do operador de recombinação entre dois indivíduos, é produzida nova descendência. Estes indivíduos, poderão ainda ser alterados com recurso ao operador de mutação. A forma como se define a nova população pode variar. Se todos os indivíduos forem substituídos por novas soluções, o processo designa-se por geracional ou técnica de sobreposição. Se apenas um grupo de indivíduos for substituído, o processo designa-se por estado estável ou técnica de não sobreposição, quando este grupo não substituído for selecionado com base nas melhores avaliações da geração anterior designa-se de elitismo. Com o passar das iterações tendem a ser geradas cada vez melhores indivíduos até atingirem um ótimo local ou global. Ótimos locais são as melhores soluções numa certa região do domínio enquanto o ótimo global é a melhor solução possível (em todo o domínio).

Para uma visão mais simplificada apresentamos o pseudocódigo do algoritmo genético:

```
Função algoritmoGenetico(problema): indivíduo
    1. Gerar população inicial
    2. Avaliar população inicial
    3. Enquanto condição de término não se verificar:
        a. Selecionar indivíduos para reprodução segundo um critério
        b. Aplicar operador de recombinação
        c. Aplicar operador de mutação
        d. Substituir a população antiga pela descendência criada
        e. Avaliar nova população
       Fim enquanto
    4. Devolve o melhor indivíduo da população final
Fim função
```

Figura 1 - Pseudocódigo do algoritmo genético

Existem múltiplas possibilidades para definir os operadores de seleção, recombinação e mutação. No Capítulo 6 é descrita a representação de indivíduos adotada e os operadores aplicados neste projeto.

3.2. Algoritmos de procura exaustiva

A procura de soluções de problemas foi uma das primeiras áreas de teste para os investigadores em inteligência artificial. Inicialmente existia a forte convicção de que um programa inteligente

seria capaz de resolver problemas através de uma abordagem de força bruta, em que o programa tentaria todas as possíveis vias para solucionar o problema considerado. Esta ideia tomou forma nos chamados algoritmos de procura.

O processo de procura resulta num conjunto de estados, que são obtidos através da aplicação de uma sequência de ações ao estado inicial. Aos estados resultantes, chamamos de espaço de estados ou espaço de procura. Para descrever um espaço de estados pode-se recorrer a um grafo, que é uma estrutura de dados composta por uma rede de arcos e nós.

Para que um agente de procura possa resolver problemas, tem de ter um conjunto de capacidades, nomeadamente: ser capaz de percecionar os estados e construir uma representação interna (um modelo) desse estado; ter capacidade de atuar sobre eles, de acordo com as regras do problema, provocando transições entre estados, o que implica ter operadores de mudança de estado; ter capacidade de reconhecer que alcançou um estado final.

A estratégia genérica, que o agente de procura adotar baseia-se na navegação pelo espaço de estados, procurando um caminho que ligue o estado inicial ao estado final. Esta estratégia é conhecida como paradigma da procura no espaço de estados. É natural que se opte por uma estratégia que satisfaça as seguintes características: ser completa, ou seja, se existir uma solução esta deve ser encontrada em tempo finito; ser discriminatória, encontrando a melhor solução caso existam várias soluções que encontre a melhor; ser económica, encontrando a solução o mais rapidamente possível e ocupando o menos espaço possível na memória [13].

O processo de procura constrói uma árvore, sendo a raiz o estado inicial e as folhas, os nós (estados) que não foram expandidos ou não têm sucessores. O conjunto de nós que ainda não foram expandidos é designado fronteira. Existem vários métodos de procura e estes diferenciam-se na estratégia de seleção do próximo nó (estado) a ser expandido.

A diferença entre os algoritmos de procura reside na ordem pela qual são adicionados os sucessores à fronteira, isto é, a ordem que eles são expandidos. No algoritmo de custo uniforme a fronteira é ordenada pelo custo do caminho desde o nó inicial ao nó em questão; no algoritmo *greedy best-first search* a fronteira é ordenada pela distância estimada do nó atual até ao nó objetivo; no algoritmo A* a fronteira é ordenada pelo resultado da adição do custo do nó inicial ao atual com a distância estimada do nó atual ao objetivo.

Na Figura 2 é demostrada a estrutura geral do algoritmo que implementa qualquer estratégia baseada no paradigma da procura num espaço de estados.

Função graph_search(problema) devolve solução ou falha

1. Inicializar a fronteira com o no 0 que contem o estado inicial do problema
2. Inicializar os conjuntos de nos explorados para estar vazio
3. Repete
    a. Se a fronteira estiver vazia então devolve falha
    b. Remover o primeiro no, *n*, da fronteira
    c. Se n contiver o estado objetivo então devolve a solução correspondente obtida através do caminho do *n* até ao nó inicial.
    d. Adicionar o estado contido em n ao conjunto de nós explorados
    e. Expandir *n* e adicionar os seus sucessores à fronteira de acordo com a estratégia do método de procura, se e só se, não existir já na fronteira ou no conjunto de nós explorados.

    Fim repete

Fim função

Figura 2 - Pseudocódigo algoritmo de procura

4. Metodologias de desenvolvimento

Tendo em conta que este projeto conta com *feedback* do cliente por intermédio do *product owner* e que os requisitos não eram estáticos, decidiu-se pela adoção de metodologias de desenvolvimento de *software* ágeis. No seguimento desta escolha, foram aplicados alguns conceitos associados à metodologia *Scrum* em conjunto com *Kanban* e *Extreme Programming* no desenvolvimento do projeto [14,15,16]. Estas metodologias promovem um conjunto de práticas que permitem uma melhor organização, divisão e evolução do trabalho com o objetivo de maximizar o valor da aplicação implementada, e principalmente adaptam-se às constantes mudanças nos requisitos. Isto significa que, em caso de alteração de funcionalidades ou ideias, é possível uma adaptação fácil sem comprometer o trabalho até então realizado.

A metodologia *Scrum* assenta na ideia de existir uma equipa com vários papéis distintos. Seguindo este pressuposto, o papel de *product owner* foi desempenhado pelo Professor Carlos Grilo, a Professora Eunice Oliveira e o Professor Luís Neves, que determinavam as tarefas a realizar. A equipa de desenvolvimento somos nós, cujo papel é executar as tarefas e concluir um potencial incremento do produto. O papel de *scrum master* foi desempenhado pelo Professor Carlos Grilo que teve como função ajudar, interagir e integrar toda a equipa.

O quadro *Kanban*, com recurso à ferramenta online *Trello*, como se pode observar na Figura 3, apresenta algumas colunas para colocar os diferentes requisitos de acordo com o seu estado de desenvolvimento, permitindo uma melhor organização do trabalho desenvolvido durante a execução do projeto. Estas colunas são: o *product backlog*, que contém os requisitos funcionais e não funcionais a implementar, o *sprint backlog*, onde se encontram as tarefas a realizar durante a sprint, a coluna *in progress*, que contém o trabalho que está a ser desenvolvido de momento, os *tests*, onde se encontram os testes a realizar para verificar a existência de erros nas funcionalidades recentemente desenvolvidas e, por fim, a coluna *done*, que contém as tarefas terminadas, da *sprint* atual. Finda a validação da *sprint*, são construídas as colunas *"Done- Sprint -i* e *"Tests- Sprint -i"*, sendo que *i*, corresponde ao número da *sprint*, onde são colocadas as tarefas validadas e terminadas. Ambas as colunas contêm as datas de início, fim e validação.

Figura 3 - Representação do quadro Kanban

Foi estabelecido que as *sprints* durariam duas semanas de forma a ser possível desenvolver as funcionalidades propostas e entregar incrementos do produto com sucesso. Portanto, na semana de início da *sprint*, era feito o planeamento do que seria executado nos 15 dias seguintes, i.e., *sprint planning.* Em algumas *sprints*, a equipa de desenvolvimento chegou a acordo de forma a colocar em prática uma das estratégias do *extreme programming* que se designa por *pair programming*. Esta estratégia é útil, na medida em que possibilita aumentar a eficácia do processo de desenvolvimento, com menor probabilidade de cometer erros ou necessidade de reformulação posterior, resultando num aumento de qualidade do código. Também encoraja uma comunicação mais próxima, melhor sintonia e noção do que cada um faz. Findada a *sprint*, decorria o *sprint review/demo* e posterior validação por parte dos docentes dando início ao planeamento da próxima.

5. Arquitetura de software

Neste capítulo, mais direcionado para a área de engenharia de software, apresentamos os requisitos funcionais (Secção 5.1), as escolhas a nível de padrões de software com recurso a um diagrama de implementação para melhor compreensão (Secção 5.2) e as tecnologias utilizadas (Secção 5.3).

5.1. Requisitos de alto nível

US1: Como utilizador pretendo importar um layout de armazém existente num ficheiro de forma a poder carregá-lo em memória e criar uma estrutura de dados para o representar.

US2: Como utilizador pretendo importar os *picks* existentes num ficheiro para carregá-los em memória e para poder representá-los graficamente.

US3: Como utilizador pretendo observar na *graphical user interface (*GUI) o *layout* do armazém e *picks* importadas para verificar que estão de acordo com as importações efetuadas.

US4: Como utilizador pretendo editar parâmetros do algoritmo genético para posteriormente o poder executar.

US5: Como utilizador pretendo executar o algoritmo genético e observar os resultados na GUI para ter uma métrica representativa da qualidade do melhor indivíduo gerado.

US6: Como utilizador pretendo executar visualmente a solução encontrada pelo algoritmo genético para verificar graficamente a melhor solução devolvida, nomeadamente os percursos dos agentes e o número de colisões.

US7: Como utilizador pretendo controlar as ações de execução visual da solução (Play/Pause), avançar e recuar para poder ter maior controlo sobre a animação e verificar as ações dos operadores.

US8: Como utilizador pretendo editar os parâmetros das experiências para posteriormente executar as mesmas.

US9: Como utilizador pretendo executar as experiências e guardar os resultados num ficheiro para posteriormente fazer uma análise dos resultados.

5.2. Diagrama de implementação

Neste projeto foi usado o padrão de apresentação *Model-View-Controller* (MVC) [17]. Este foca-se na diminuição da interdependência entre as três componentes, o modelo, a vista e o controlador. O benefício do uso deste modelo é o isolamento da lógica de negócio e da lógica de apresentação, o que possibilita que as vistas sejam alteradas sem termos de adaptar as regras de negócio, proporcionando assim flexibilidade e oportunidade de reutilização da lógica de negócio.

O modelo consiste na parte lógica da aplicação que manipula os dados através de regras de negócio, lógica e funções. O modelo notifica as vistas quando se altera e permite que um controlador aceda às suas funcionalidades. Além disso, encapsula os dados e comportamento independentemente da apresentação.

A vista apresenta os conteúdos mapeados pelo modelo e atualiza a apresentação quando o modelo se altera e efetua o direcionamento dos *inputs* do utilizador para o controlador.

O controlador recebe os pedidos do utilizador, realizados através da vista, transforma-os em ações a serem realizadas pelo modelo e seleciona a vista a apresentar ao utilizador.

O padrão MVC consiste em duas separações principais: separa a vista do modelo e o controlador da vista. A mais importante destas, é a separação entre a vista e o modelo, que torna possível o uso de múltiplas vistas com o mesmo modelo e permite que as vistas possam ser alteradas sem termos de alterar o modelo.

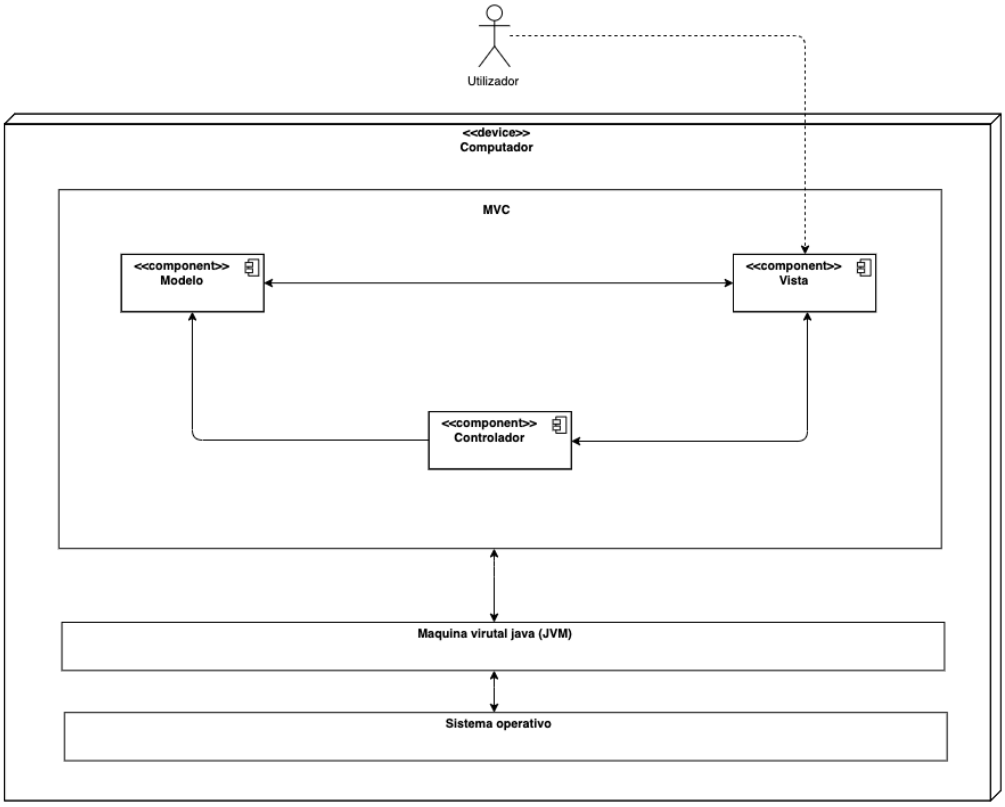


Figura 4 - Diagrama de implementação

A Figura 4 sendo um diagrama de implementação apresenta o dispositivo no qual a aplicação é executada, a interação entre o sistema operativo e a máquina virtual Java, mas também entre a *java virtual machine* (JVM) e a aplicação que segue um padrão de apresentação MVC.

5.3. Tecnologias utilizadas

Para a implementação deste projeto optou-se pela linguagem de programação Java e pela *framework* de desenvolvimento JavaFX. A escolha da linguagem foi unânime na medida em que durante a licenciatura foi utilizada em projetos de várias unidades curriculares, o que proporcionou a aquisição de conhecimentos e um elevado grau de familiaridade com a mesma e facilitou o desenvolvimento deste projeto. Inicialmente implementou-se a GUI com recurso a *Java Swing*. No entanto, verificou-se que a tecnologia não seria a mais indicada, tendo em conta que o esforço para implementar uma funcionalidade gráfica é muito maior em comparação com outras. Estas alternativas, permitem assim, alocar o tempo poupado para a implementação de algoritmos que respondem às necessidades do projeto. Assim, e após pesquisa em [19], chegou-se à conclusão de que o ideal seria a migração para uma tecnologia mais recente com melhores ferramentas de desenvolvimento. Consequentemente a escolha passou pela *framework* JavaFX, permitindo implementar certas funcionalidades e customizar com maior liberdade e facilidade.

6. Solução desenvolvida

Neste capítulo descreve-se e justificam-se as diferentes abordagens que foram adotadas. O problema a revolver (Secção 6.1), a representação do armazém (Secção 6.2), a representação do armazém em ficheiro (Secção 6.3), o algoritmo de procura A* (Secção 6.4), o algoritmo genético (Secção 6.5), que incluí a representação dos indivíduos, operadores de seleção, recombinação, mutação, a função de avaliação utilizada para avaliar o mérito de cada indivíduo, e as diferentes formas como se lida com colisões e com as restrições de peso dos produtos. É ainda explicado o porquê do uso de um algoritmo de procura em conjunto com uma meta-heurística em detrimento de apenas um algoritmo de procura. Descreve-se também o módulo que permite a realização de experiências (Secção 6.6), a independência do modelo (Secção 6.7), os testes de software (Secção 6.8) e a interface gráfica (Secção 6.9).

6.1. Problema a resolver

O problema que está a ser resolvido, consiste na distribuição de produtos (*pick*) pelos diversos operadores (agentes), para que estes possam ser recolhidos com a máxima eficiência possível. Neste projeto é adotado o sistema de *picker-to-parts* onde os agentes se deslocam pelos corredores para recolher itens de uma ou mais encomendas e é tida em consideração a capacidade de cada produto (peso que cada produto aguenta em cima), o limite de peso dos agentes e as colisões entre estes. Foi ainda considerado, que para o mesmo armazém, os diferentes produtos se encontram sempre no mesmo local, isto é, num armazém o produto x encontra-se sempre na posição y.

6.2. Representação do armazém

O armazém é representado através de um grafo. Neste grafo existem diferentes tipos de nós, os de decisão que representam os pontos onde os agentes decidem a sua próxima ação (cima, baixo, esquerda e direita), os nós *picks* (produto a recolher das prateleiras) que correspondem às posições onde os agentes podem recolher produtos, os nós que representam as posições iniciais dos agentes e o nó que corresponde ao ponto de entrega. Para cada armazém, os nós de decisão são fixos (estão sempre na mesma posição), enquanto os *picks* variam consoante a posição dos produtos da encomenda que está a ser recolhida num certo momento.

As arestas do grafo representam os corredores, podendo estes serem monoagente (não é possível o cruzamento de agentes) ou multiagente (possível o cruzamento de agentes). Na prática, isto significa que nas arestas monoagente pode haver colisões entre dois ou mais agentes quando estes se cruzam enquanto nas arestas multiagente não existem colisões.

Cada *pick* tem um peso e uma capacidade. A capacidade corresponde ao peso máximo que pode ser colocado em cima deste produto sem causar danos. É ainda considerada a capacidade de cada agente (peso que este consegue transportar).

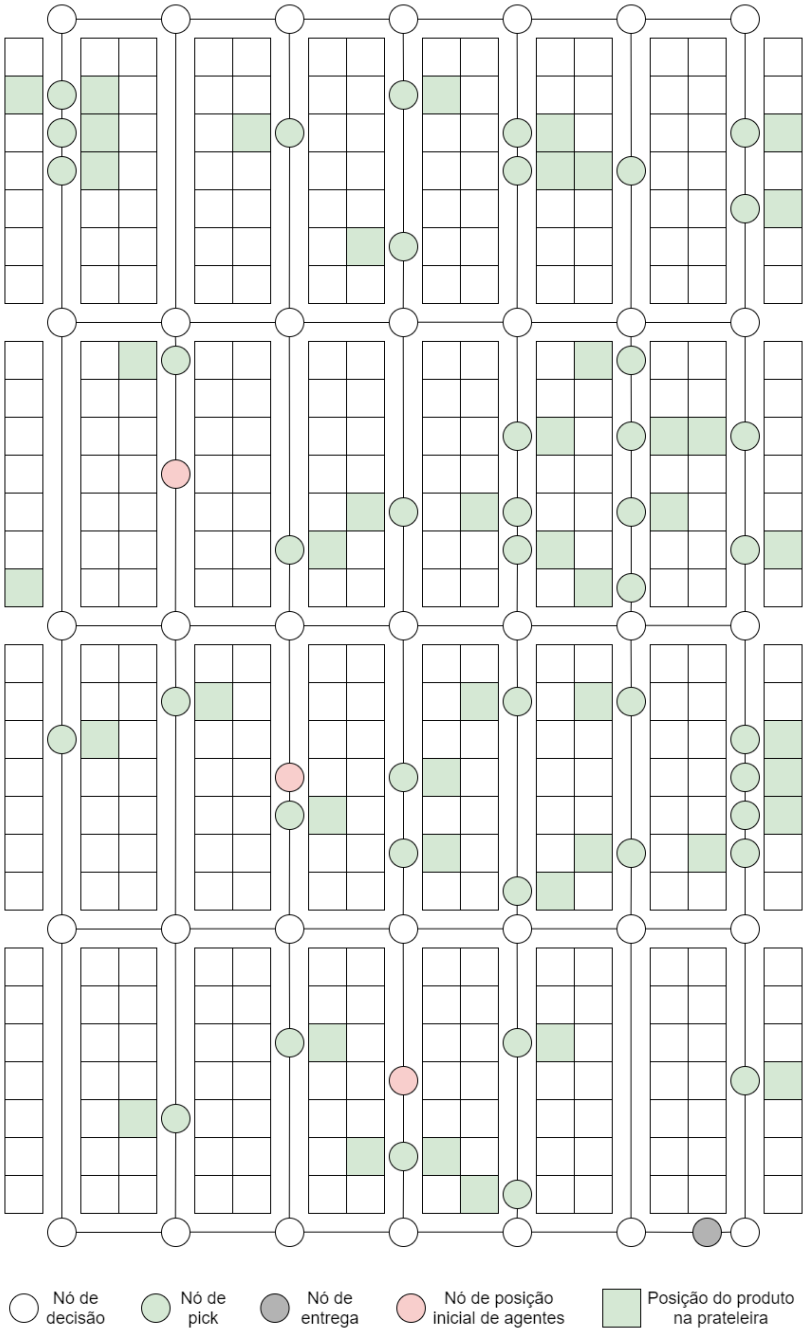


Figura 5 - Representação do armazém

Decidiu-se utilizar um grafo para representar o armazém pois este, comparado com uma matriz, permite otimizar o algoritmo de procura quanto ao seu tempo de execução e memória utilizada.

Para efeitos deste projeto considerou-se que todos os agentes se movem com a mesma velocidade constante.

6.3. Representação do armazém em ficheiro

O projeto desenvolvido permite importar diferentes disposições de armazéns, produtos e agentes, a partir de ficheiros de texto, com os dados formatados em *JavaScript Object Notation* (JSON) recorrendo à biblioteca *google/json*[1]. Usaram-se dois ficheiros: um que descreve a configuração do armazém, onde são especificados os pontos de decisão, os corredores, a posição inicial dos agentes, a posição do ponto de entrega e as dimensões do armazém; outro em que é especificada a posição dos *picks*. Estes dois ficheiros são representados na Figura 6.

É ainda, validada a estrutura destes documentos através de um JSON *schema*, que verifica se estes documentos contêm os campos necessários. Para isto é usada a biblioteca *everit-org/json-schema*[2].

Figura 6 - Representação do ficheiro do armazém (esquerda) e do ficheiro dos *picks* (direita)

[1] https://github.com/google/gson
[2] https://github.com/everit-org/json-schema

6.4. A*

Para calcular a distância e o caminho entre dois nós, foi usado o algoritmo de procura A*, visto que este é completo (sempre que existe uma solução, esta é encontrada). Se a função de heurística for admissível este é também ótimo (encontra sempre a melhor solução). Tentou-se minimizar a complexidade espacial do algoritmo utilizando um grafo para representar o armazém, pois esta abordagem apresenta uma quantidade menor de posições possíveis para os agentes quando comparada com uma matriz para representar o armazém, fazendo assim com que o grafo de nós gerado pelo A* tenha menos níveis, o que na prática representa uma otimização significativa.

O custo de cada nó n gerado pelo A* é estimado através de $f(n) = g(n) + h(n)$ em que $g(n)$ representa o custo do caminho desde o nó inicial até ao nó n e $h(n)$ representa a função heurística, que neste problema é a distância de M*anhattan* entre n e o nó objetivo. A distância M*anhattan* é uma heurística aceitável pois esta nunca superestima a distância ao objetivo. Isto é, o valor da heurística é sempre menor ou igual ao custo real do menor caminho de n ao objetivo. Esta distância M*anhattan* satisfaz ainda a condição de consistência em que para todos os nós n e todos os seus sucessores, o custo estimado para atingir o objetivo a partir de n é menor ou igual que o custo real de ir de n até um dos seus sucessores mais o custo estimado para chegar ao objetivo a partir deste sucessor. Neste projeto, em vez de calcularmos *a priori* todos os caminhos possíveis entre nós, estes são calculados e guardados apenas quando são necessários, evitando assim a realização de operações desnecessárias.

6.5. Algoritmo genético

Tendo em conta que este projeto pretende lidar com armazéns de tamanho considerável e que os recursos computacionais são limitados, optou-se por usar uma abordagem em que se combina o uso de um algoritmo genético com um algoritmo de procura exaustiva em vez de se usar apenas o algoritmo de procura exaustiva. Apesar da abordagem que recorre apenas ao algoritmo de procura permitir obter sempre a melhor solução (o que não acontece com a abordagem escolhida) o tempo e a memória usados para a obter exclusivamente por essa via seriam muito superiores, tornando inviável essa abordagem. Depois de estudar este problema decidiu-se que o compromisso de não ser possível garantir que se encontra a melhor solução era aceitável tendo em conta a diminuição drástica no tempo de execução.

6.5.1. Representação dos indivíduos

Um indivíduo representa a distribuição dos *picks* pelos vários agentes bem como a ordem pela qual cada um recolhe os seus *picks*. Como se pode observar na Figura 7, um indivíduo da população é caracterizado por um vetor de números inteiros que representa o genoma. O tamanho é o resultado da soma do número de *picks* com o número de agentes, menos um (o último agente não precisa de identificação). Cada valor superior a zero que se encontra no genoma corresponde à posição (índice) de um *pick* na lista de *picks*. Os valores inferiores a zero separam os *picks* para cada agente.

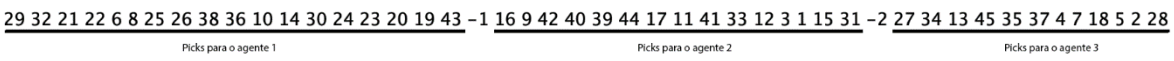


Figura 7 - Representação de um indivíduo

Para compreender que objetos dizem respeito a cada agente e quando o número de agentes é superior a um, percorre-se o vetor começando no índice zero até encontrar um número negativo, significando assim, que os valores atrás encontrados dizem respeito aos *picks* do agente 1. Partindo da posição seguinte ao valor negativo encontrado, procura-se outro número inferior a zero ou o fim do vetor, significando que foram encontrados os pontos de recolha para o próximo agente e assim sucessivamente. Caso apenas exista um agente, este é responsável por recolher todos os *picks*.

6.5.2. Função de avaliação

Os algoritmos genéticos, têm como objetivo gerar progressivamente melhores soluções para o problema a que estão a ser aplicados. Para isto, é necessário avaliar a qualidade de cada indivíduo e atribuir-lhe um valor (*fitness*) que quantifica a qualidade da solução. Neste projeto esta avaliação é influenciada por quatro parâmetros: o tempo, as colisões, a capacidade dos produtos e a capacidade dos agentes.

Cada indivíduo representa n agentes e o seu percurso. A partir desta representação, recorrendo ao algoritmo de procura A*, conseguimos determinar a distância que cada agente tem de percorrer até chegar ao ponto de entrega, dando assim por completa a sua atividade para a encomenda atual. Como a velocidade de todos os agentes é igual e constante (a aceleração é zero), pode-se usar a distância para representar o tempo que cada um demora a concluir as suas

recolhas. Não tendo em conta as colisões e sendo Δt o intervalo de tempo, d a distância e v a velocidade, analisando a equação $\Delta t = \frac{d}{v}$ que define o tempo em função da velocidade e da distância, chega-se à conclusão de que, quando a velocidade é constante, o tempo é influenciado apenas pela distância percorrida (o tempo é proporcional à distância). Depois de determinada a distância que cada agente percorre, atribui-se ao parâmetro tempo do *fitness* o maior valor obtido. Foi escolhida esta abordagem, uma vez que se pretende minimizar o tempo para recolher todos os produtos e colocá-los na área de recolha. Na prática, isto significa que o algoritmo vai tentar distribuir os *picks* de tal forma que todos os agentes percorram aproximadamente a mesma distância. Outra abordagem que teoricamente pode parecer apelativa, seria usar a soma do tempo de todos os agentes para a componente tempo da *fitness*, mas analisando esta abordagem, rapidamente se percebe que esta, poderia gerar soluções em que um agente faria todos os *picks*, o que na prática seria muito ineficiente, pois significa desperdiçar recursos em agentes que estariam inativos e a encomenda demoraria muito mais a chegar ao ponto de entrega.

Numa ambiente multiagente existe a possibilidade de haver colisões em corredores monoagente ou em pontos de decisão. Assim sendo, foi necessário tê-las em conta, através do parâmetro colisões da função de avaliação. Neste projeto foram implementadas três formas de determinar o valor deste parâmetro, que são analisadas e comparadas nos parágrafos seguintes.

A primeira abordagem para lidar com colisões é a mais simples. Nesta, atribui-se uma ponderação ao tempo e ao número de colisões, definindo assim o peso que cada um destes parâmetros têm na *fitness*. Apesar da sua simplicidade, esta abordagem pode não ser a mais adequada, pois não reconhece diferenças entre colisões. Isto é, na prática, existem colisões que podem ser resolvidas com muito pouco esforço por parte dos agentes, enquanto outras requerem mais algum esforço e principalmente mais tempo. Com esta abordagem a *fitness* é determinada segundo a seguinte equação, $fitness = (wT * tempo) + (wC * colisoes)$ onde $wT\ e\ wC$ correspondem às ponderações do tempo e das colisões, respetivamente, $tempo$ corresponde ao tempo máximo que os agentes demoram a recolher os produtos (descrito antes), sem ter em conta as colisões e $colisoes$ corresponde ao número total de colisões do indivíduo.

A segunda e terceira abordagens para lidar com colisões, são mais específicas e lidam com cada colisão individualmente. Estas, determinam o tempo que é gasto para evitar a ocorrência de cada colisão e atribuem este tempo ao parâmetro colisões. Isto é, para que uma colisão entre dois agentes não ocorra, um terá que esperar num ponto de decisão para deixar o outro passar.

A diferenças entre estas duas abordagens, é a maneira como escolhem o agente que espera, para deixar o outro passar. Na segunda abordagem, o agente que espera é sempre aquele que se encontra mais perto de um ponto de decisão relativamente ao ponto de colisão ou, por outras palavras, o que tem de esperar menos para resolver a colisão. Na terceira abordagem, soma-se ao tempo que cada agente envolvido na colisão demora a realizar os seus *picks* (incluindo o tempo despendido na resolução de colisões prévias), o tempo que este tem que esperar no ponto de decisão mais próximo para deixar o outro agente envolvido na colisão passar e define-se que o agente que espera, é aquele que apresenta o menor valor. Isto é, espera o agente que, depois de resolvida a colisão, apresente o menor tempo para chegar ao destino. Com esta abordagem pode acontecer que o agente que espera é aquele que se encontra mais longe de um ponto de decisão.

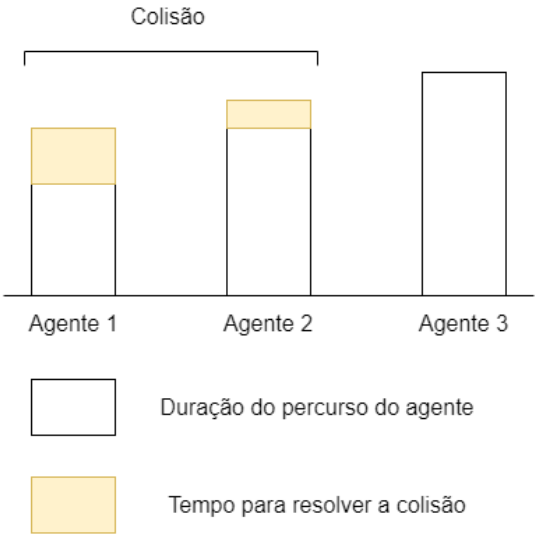


Figura 8 - Representação de uma colisão entre dois agentes

Tendo em conta o exemplo da Figura 8, pode-se ilustrar o funcionamento da segunda e terceira abordagens para tratar colisões. Aplicando a segunda abordagem, o agente que espera é o 2 pois é o que apresenta um menor custo para o fazer. Usando a terceira abordagem, apesar do agente 1 se encontrar mais distante do ponto de decisão, é adicionado este valor ao parâmetro colisões pois este agente apresenta um valor menor, quando é adicionado o tempo que demora a fazer as suas recolhas e a resolver esta colisão. Devido à natureza destas duas abordagens e às características do armazém usado nas experiências realizadas, para a segunda abordagem o tempo máximo de espera para resolver uma colisão, é cinco, visto que os corredores têm um comprimento de 7 unidades, e que esta abordagem escolhe sempre o agente que tem de esperar um menor tempo. A colisão que requer um maior tempo de espera ocorre, a meio do corredor.

A partir desta posição é necessário percorrer 4 unidades para chegar ao ponto de decisão, mais uma unidade, que corresponde ao impedimento deste nó como é demostrado na Figura 9. Na terceira abordagem o tempo máximo de espera é 8 pois a colisão pode ocorrer numa das pontas do corredor e o agente pode ter que esperar junto ao ponto de decisão da ponta oposta. Ambos os casos são ilustrados na Figura 9.

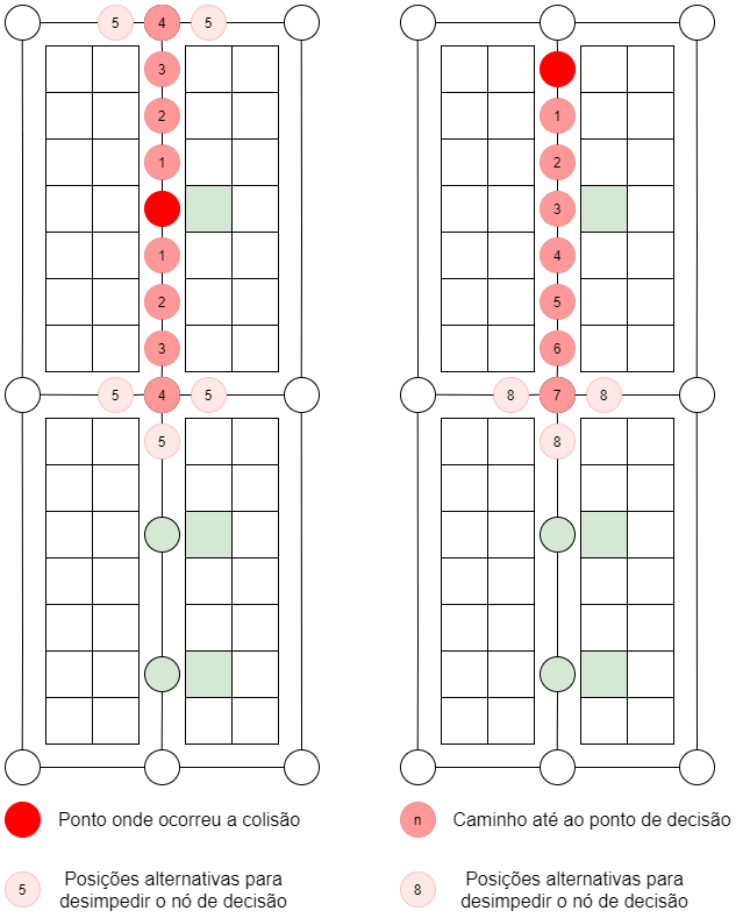


Figura 9 - Tempo máximo para resolver uma colisão usando a segunda abordagem (esquerda) e usando a terceira abordagem (direita)

Tanto a segunda como a terceira abordagem atribuem ao parâmetro colisões, o tempo gasto em todas as colisões do indivíduo e calculam a *fitness* de acordo com a seguinte equação, $fitness = tempo + colisoes$ em que $tempo$ corresponde ao tempo máximo que os agentes demoram a recolher os produtos, sem ter em conta as colisões e $colisoes$ corresponde ao tempo despendido a resolver todas as colisões do indivíduo.

Por último, os produtos que estão a ser recolhidos pelos diversos agentes, têm uma certa capacidade (peso que suportam em cima). É importante referir que em certos contextos a

danificação de produtos é inadmissível, logo tem de ser evitada a todo o custo, ao contrário das colisões, que se pretendem minimizar, mas são admissíveis, uma vez que no projeto se considera que os operadores são humanos e podem evitar as colisões. Esta diferença, é a razão pela qual não foi adotada uma abordagem que permita que se desrespeite a restrição relativa à capacidade dos produtos. Assim, sempre que um agente se desloca para recolher o próximo produto, verifica-se se com a adição deste, a capacidade de algum dos produtos recolhidos anteriormente é excedida. Se sim, em vez do agente ir buscar o próximo produto, este vai deixar os produtos recolhidos anteriormente ao ponto de entrega e de seguida continua o seu percurso. Caso contrário, o agente continua a recolher produtos, até se deparar com restrições de capacidade ou até não ter mais produtos para recolher. Esta abordagem, apesar de não contribuir diretamente para um parâmetro da função de avaliação, contribui indiretamente para o parâmetro tempo, pois a deslocação ao ponto de entrega vai inevitavelmente aumentar a distância percorrida pelo agente. Visto que, com o decorrer das gerações do algoritmo genético, este parâmetro (tempo) tende a diminuir, é razoável assumir, que o número de vezes que os agentes se deslocam ao ponto de entrega, vai também ser reduzido através de uma melhor distribuição dos *picks* pelos agentes, não só quanto à distância percorrida, mas também, a colocação de produtos com maior capacidade e/ou mais pesados na base do agente (recolhidos primeiro).

6.5.3. Operadores de seleção

Os operadores de seleção são necessários para implementar corretamente o algoritmo genético uma vez que definem a forma como são selecionados os indivíduos que vão produzir a nova geração. Este procedimento possibilita que os melhores indivíduos apareçam com maior frequência na população seguinte e consequentemente se reproduzam mais vezes. Se a seleção for muito exigente, ou seja, os melhores indivíduos são sempre os escolhidos, pode levar à estagnação num máximo local. No entanto, se for pouco exigente pode levar a um processo de evolução muito lento.

Neste projeto foram implementados dois métodos de seleção de referência, o de seleção por torneio e o método baseado em ordenação. Ambos os operadores permitem evitar que um pequeno número de indivíduos, muito melhores que o resto, tomem controlo da população, fazendo com que partes do domínio fiquem por explorar.

Para executar o operador de **seleção por torneio** é necessário que o utilizador defina na interface o tamanho do torneio, isto é, o número de indivíduos que serão comparados em cada

iteração de forma a eleger o melhor e assim, adicionar à nova população. Com o intuito de manter o mesmo tamanho da população, o processo a seguir descrito é repetido um número de vezes igual ao número de indivíduos da população. A Figura 10 pretende ilustrar funcionamento do operador.

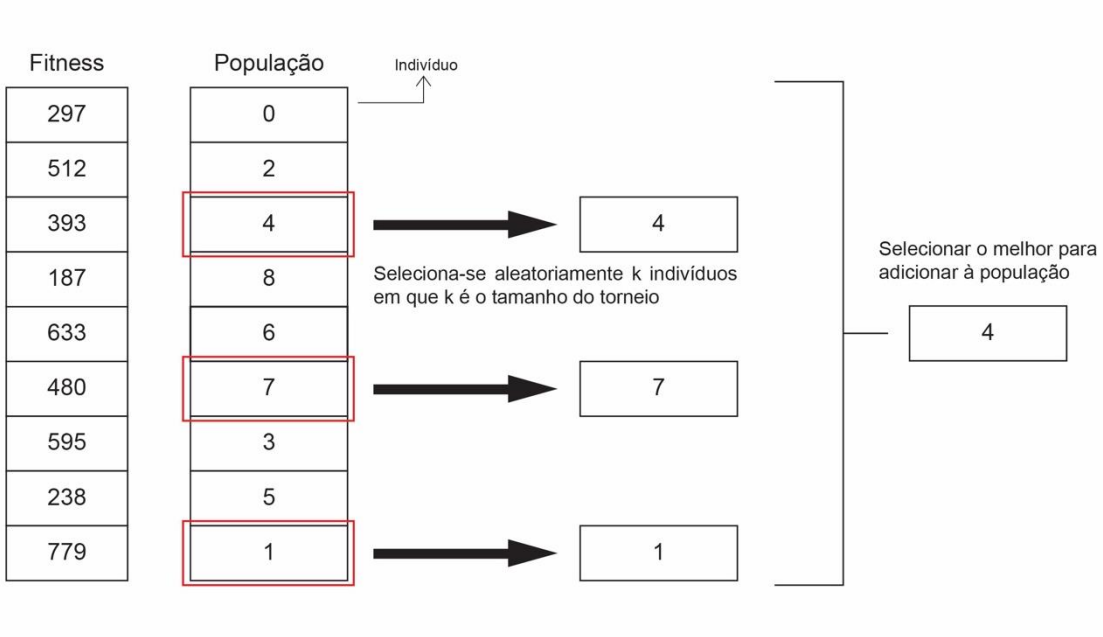


Figura 10 – Método de seleção por torneio

Como se pode observar na figura acima, selecionam-se *k* indivíduos, sendo *k* o tamanho do torneio e escolhe-se o melhor para adicionar à nova população. A comparação é efetuada recorrendo ao valor devolvido pela função de avaliação (*fitness*) quando aplicado a cada indivíduo. Neste caso, o melhor indivíduo é o que tem a fitness mais baixa entre os que participam no torneio.

O método de **seleção baseado em ordenação** é computacionalmente menos eficiente do que o anterior, contudo não significa que obtenha piores resultados uma vez que procura reduzir a convergência para um mínimo local.

A Figura 11 pretende demonstrar como o operador de seleção baseado em ordenação pode ser representado. A presente tabela contém a posição de cada indivíduo na população, o seu valor de mérito, a probabilidade de cada indivíduo ser escolhido tendo em conta o valor da pressão seletiva introduzida pelo utilizador na interface gráfica e as probabilidades acumuladas também representadas recorrendo a uma linha dividida em intervalos.

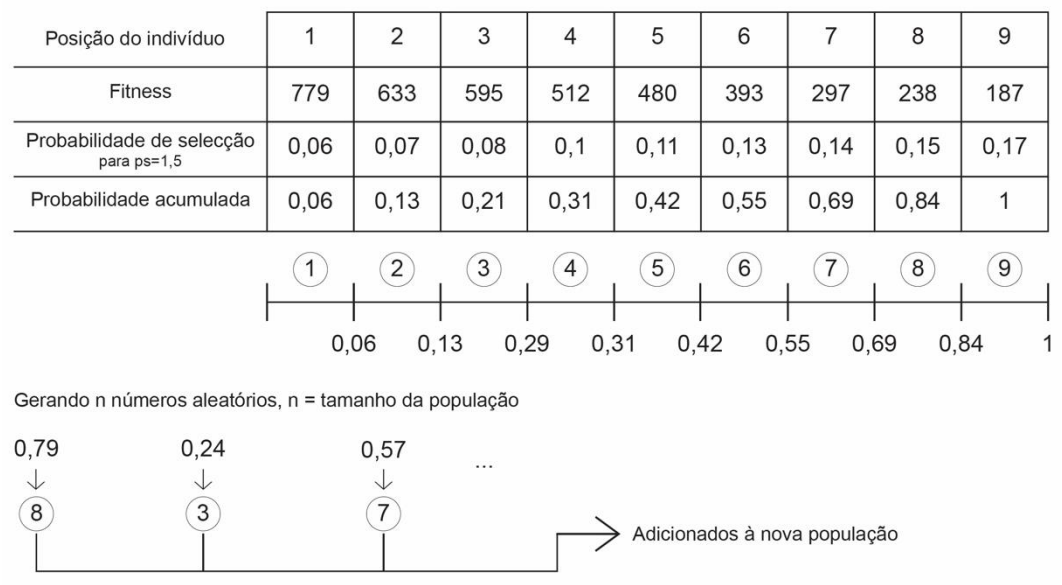


Figura 11 - Seleção baseada em ordenação

Neste operador, os indivíduos são ordenados de acordo com a sua *fitness* sendo que o melhor indivíduo é colocado na posição N (tamanho da população) e o pior na posição 1. Neste problema queremos minimizar a *fitness* e, desta forma, o melhor indivíduo é aquele que tem o resultado da função de mérito mais baixa. O próximo passo consiste em atribuir a cada indivíduo, a probabilidade de ser escolhido usando uma distribuição linear em que a soma das probabilidades é igual a um. A probabilidade de cada indivíduo i ser escolhido é dada pela fórmula: $p_i = \frac{1}{N}\left[2 - PS + 2(PS-1)\frac{i-1}{N-1}\right]$, onde $PS \in [1,2]$ que representa a pressão seletiva. A pressão seletiva pode ser vista como um valor que mede a influência do meio podendo gerar uma implementação mais realista tendo em conta a teoria da evolução de Darwin. Quanto maior o valor da pressão seletiva mais o método de seleção tem em conta a posição do agente no *ranking*, levando a uma maior tendência à exploração de soluções já existentes e maior a velocidade de convergência. Por outro lado, valores mais baixos tendem a levar a uma melhor pesquisa do espaço de soluções, aumentando a confiabilidade da solução indicada como ótima.

Após obter as probabilidades de cada indivíduo ser selecionado, guardam-se as probabilidades acumuladas, isto é, para cada posição do vetor de probabilidades acumuladas coloca-se o resultado do somatório das probabilidades dos indivíduos anteriores. O passo seguinte passa por recorrer ao método da roleta. Assim, é gerado um valor aleatório entre 0 e 1 e o primeiro indivíduo cuja probabilidade acumulada ultrapasse este valor é aquele escolhido para ser

adicionado à nova população. Este processo da roleta é efetuado o número de vezes igual ao tamanho da população.

6.5.4. Operadores de recombinação

Os indivíduos permutam o seu material genético, com uma probabilidade que é definida na interface gráfica pelo utilizador e que normalmente, varia entre 0.5 e 0.8, produzindo indivíduos com novas características. O operador de recombinação, utiliza dois progenitores da população gerada, selecionados por um dos métodos de seleção anteriormente descritos. Para decidir quais os pares de indivíduos que serão usados para este processo, é gerado um valor aleatório para ser comparado com a probabilidade que o utilizador escolheu inicialmente. Caso esta probabilidade seja superior ao valor aleatório gerado, os dois primeiros indivíduos são escolhidos para produzir descendência. Após a produção de descendência, são escolhidos os dois próximos indivíduos da população e este processo repete-se até se chegar ao fim da população. A forma como o material genético é trocado depende do operador a utilizar. Neste projeto implementaram-se quatro operadores de referência sendo estes: *Partially Mapped Crossover* (PMX), *Cycle crossover (*CX), *Order crossover* (OX) e *Order crossover 1* (OX_1).

A Figura 12 é uma representação do operador **PMX** em que os dois progenitores p_1 e p_2 vão dar origem aos descendentes f_1 e f_2 e o mapeamento entre os elementos da área de corte de cada um dos pais.

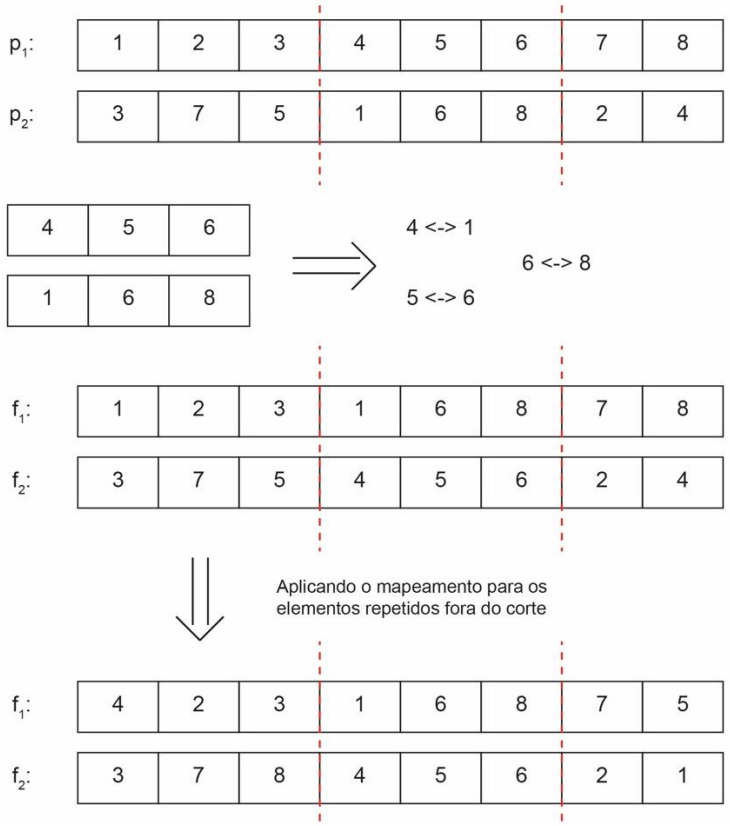


Figura 12 - Representação do método de recombinação PMX.

O operador permite criar os descendentes selecionando dois pontos de corte aleatórios para aplicar em ambos progenitores. Supondo que este corte é entre o índice i e j, admitindo que, $j > i$, resulta: [1 2 3|4 5 6|7 8] e [3 7 5|1 6 8| 2 4]. Os elementos da área de corte são mapeados da seguinte forma, $4 \leftrightarrow 1$, $5 \leftrightarrow 6$ e $6 \leftrightarrow 8$. O corte do primeiro progenitor (p_1) é copiado para o segundo descendente (f_2) e o do segundo progenitor (p_2) para o primeiro descendente (f_1). Como resultado obtém-se, f_1: [$x\ x\ x$|1 6 8|$x\ x$] e f_2: [$x\ x\ x$|4 5 6|$x\ x$]. De seguida, os restantes elementos são copiados do p_1 para o f_1 e igual para p_2 e f_2 . Caso os elementos copiados fora do corte já existam no descendente, estes são trocados de acordo com o mapeamento inicial. Isto é, o primeiro elemento do f_1 deve tomar o valor 1, como o primeiro elemento do seu progenitor. No entanto, como este valor já existe, é substituído pelo seu mapeamento $1 \leftrightarrow 4$. Desta forma, o primeiro elemento do f_1 será 4. Os restantes elementos podem ser copiados sem problema com a exceção do último que é o 8. Como este já existe, é utilizado o mapeamento $8 \leftrightarrow 6$, e como 6 também já existe, $6 \leftrightarrow 5$, sendo o 5 escolhido para ocupar a última posição do primeiro progenitor. Fazendo o mesmo processo para o f_2 resultam os seguintes descendentes: f_1 : [4 2 3|1 6 8| 7 5] e f_2 : [3 7 8| 4 5 6| 2 1].

O operador **OX,** como pode ser observado na Figura 13, necessita de dois progenitores p_1 e p_2 representados através de vetores de inteiros que vão dar origem aos descendentes f_1 e f_2.

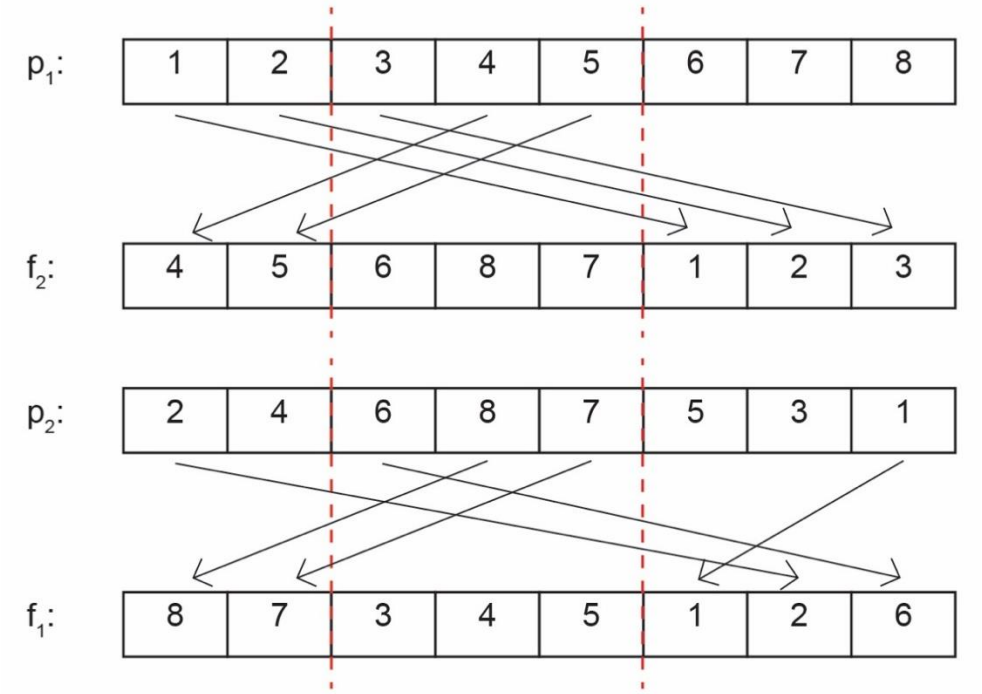


Figura 13 - Representação do operador OX

Para construir um descendente, começa-se por determinar uma área de corte a efetuar em ambos os progenitores. Considerando, p_1: [1 2 3 4 5 6 7 8] e p_2 : [2 4 6 8 7 5 3 1]. Supondo que se seleciona o corte entre os índices i e j, admitindo que $j > i$, em ambos os progenitores obtemos, p_1 : [1 2|3 4 5| 6 7 8] e p_2 : [2 4|6 8 7|5 3 1]. De seguida, são criados os descendentes que apresentam uma cópia da área de corte, p_1 gera f_1 : [$x\ x$|3 4 5|$x\ x\ x$] e p_2 gera f_2 : [$x\ x$|6 8 7|$x\ x\ x$]. Partindo do segundo ponto de corte, j, de um dos progenitores (p_1), os restantes elementos são copiados sequencialmente para o descendente do outro progenitor (f_2) começando pela posição seguinte a j, sem colocar os valores que já existem. No caso de se verificar a existência de algum elemento repetido, seleciona-se o elemento da posição seguinte e assim em diante. Depois de chegar ao fim do vetor do progenitor (p_1), retorna-se ao início (posição 0) para serem adicionados os restantes elementos em falta ao descendente (f_2) resultando em, f_1 : [8 7|3 4 5| 1 2 6] e f_2 : [4 5|6 8 7| 1 2 3].

O operador **OX1** é uma variante do anterior OX e a Figura 14 é representativa do algoritmo seguindo a mesma estrutura do operador anterior.

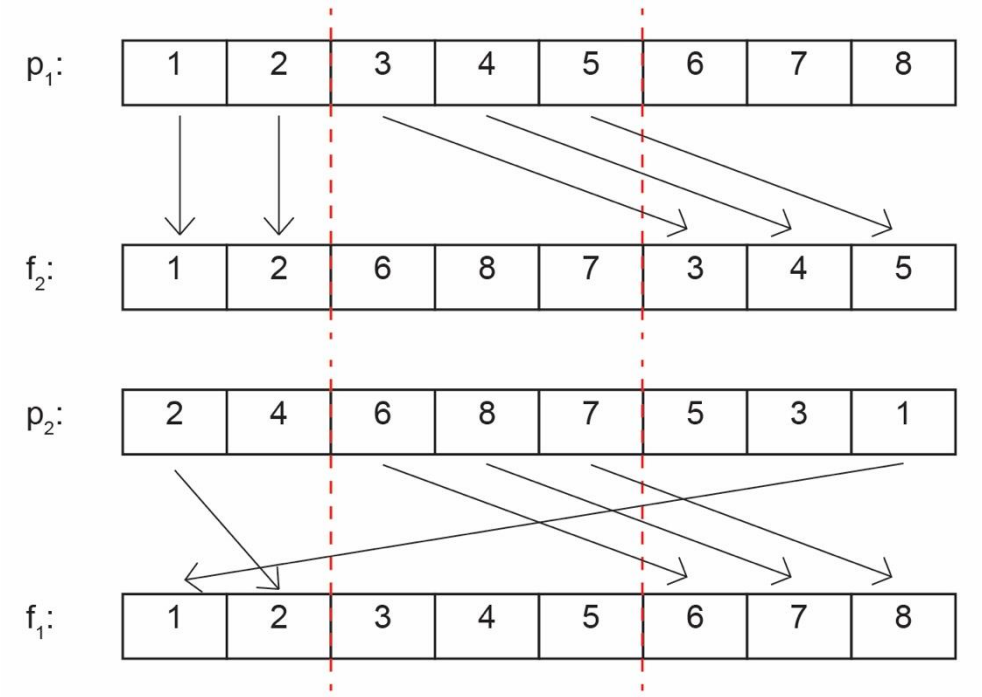


Figura 14 - Representação do operador *OX*1

A diferença entre estes dois operadores reside na forma como são adicionados os restantes elementos a um descendente depois da criação dos progenitores através da cópia das áreas de corte, isto é, p_1 gera f_1 : [x x|3 4 5|x x x] e p_2 gera f_2 : [x x|6 8 7|x x x]. Assim, começando na posição a seguir ao último ponto corte do progenitor (p_1), colocam-se os valores em falta no vetor do descendente criado pelo outro progenitor (f_2) partindo da posição inicial deste vetor (e não a seguir ao segundo ponto de corte do descendente como no OX) sem colocar valores repetidos. A grande diferença é que o operador OX, para progenitores iguais cria descentes iguais a estes enquanto o OX1 cria descendentes diferentes. Esta particularidade do operador OX, resulta numa diminuição da diversidade da população, o que em certos casos pode levar a áreas de domínio que fiquem por explorar.

O operador **CX** tem como resultado a criação de dois descendentes a partir de dois progenitores. A Figura 15 é uma representação do operador que contém os progenitores p_1 e p_2. Por questões de formatação do documento apenas é considerado um descendente f_1 e os elementos resultantes dos ciclos obtidos da aplicação do algoritmo.

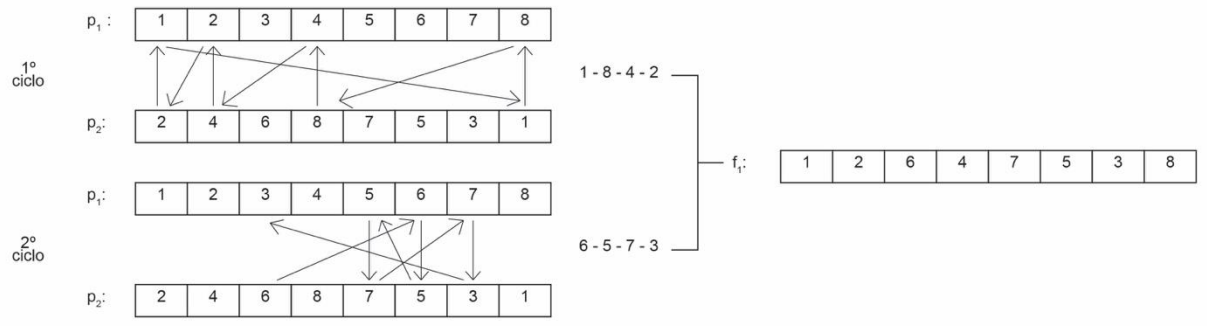


Figura 15 - Representação do operador CX

Dados dois indivíduos progenitores, p_1 : [1 2 3 4 5 6 7 8] e p_2 : [2 4 6 8 7 5 3 1], para gerar o primeiro filho (f_1) considera-se a primeira posição de p_1 ou p_2. Considerando que é selecionado o valor 1 de p_1, insere-se na primeira posição de f_1. Visto que o elemento 1 se encontra na 8ª posição de p_2 e o elemento nesta posição em p_1 é o 8, insere-se então o número 8 na 8ª posição de f_1. De forma semelhante ao passo anterior, inserimos o valor 4 na 4ª posição de f_1 porque o valor 8 encontra-se na 4ª posição de p_2 e porque o valor nesta posição em p_1 é 4. Uma vez que na 2ª posição de p_2 encontra-se valor inserido anteriormente (4) e na mesma posição de p_1 encontra-se o valor 2, este é inserido na 2ª posição de f_1. O ciclo termina quando é alcançado novamente o primeiro valor do ciclo, isto é, o valor anteriormente inserido é o 2 que se encontra na posição 1 do indivíduo p_2 e nesta posição em p_1 encontra-se o valor 1 que já existe no descendente. O segundo ciclo deve ser iniciado a partir de p_2, começando pela primeira posição que ainda não foi explorada (neste caso responde à 3ª posição). Considerando a 3ª posição de p_2, que corresponde ao valor 6, este será colocado na posição 3 de f_1. Seguindo a mesma lógica as posições 5, 6 e 7 de f_1 devem ser preenchidas pelos valores 7, 5 e 3 respetivamente. No fim deste processo f_1: [1 2 6 4 7 5 3 8]. Para gerar f_2 o processo é similar ao descrito anteriormente trocando a ordem dos progenitores.

6.5.5. Operadores de mutação

Após gerada a descendência através dos operadores de recombinação, os operadores de mutação podem efetuar alterações num ou mais genes de um indivíduo. Estas são muito importantes porque tentam evitar que a solução estagne num mínimo local, fazendo com que explore outras regiões eventualmente mais promissoras. A probabilidade de mutação que o utilizador define normalmente é muito baixa e, da mesma forma que no processo de recombinação, é gerado um valor aleatório e caso seja inferior à probabilidade definida, é

efetuada a mutação. Neste projeto implementaram-se três operadores de referência: *Insert*, *Inversion* e *Scramble.* As figuras a seguir apresentadas têm a mesma estrutura: um indivíduo representado por um vetor de inteiros e o resultado da aplicação do operador de mutação.

O algoritmo associado ao operador ***Insert***, representado na Figura 16, seleciona dois pontos aleatórios e permuta sucessivamente o último elemento da área de corte com todos os elementos anteriores de forma a tomar a primeira posição da área de corte.

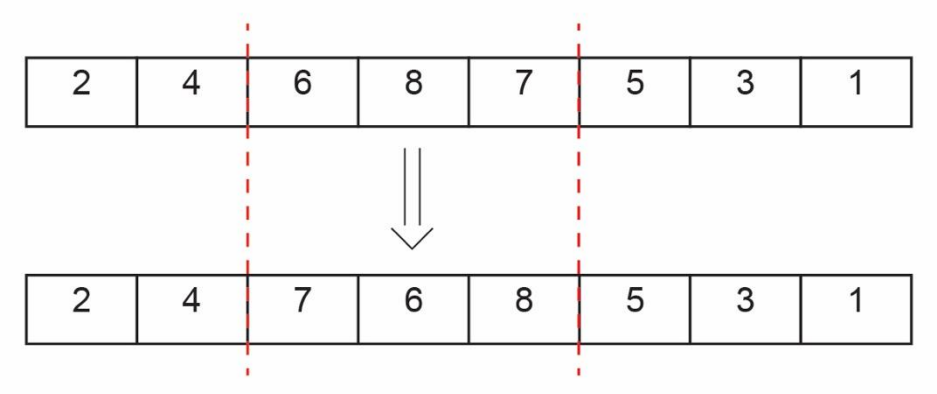


Figura 16 - Representação do operador *Insert*

O operador ***Inversion***, representado na Figura 17, consiste em selecionar dois pontos aleatórios na sequência de genes e inverter e subsequência compreendida entre estes dois pontos.

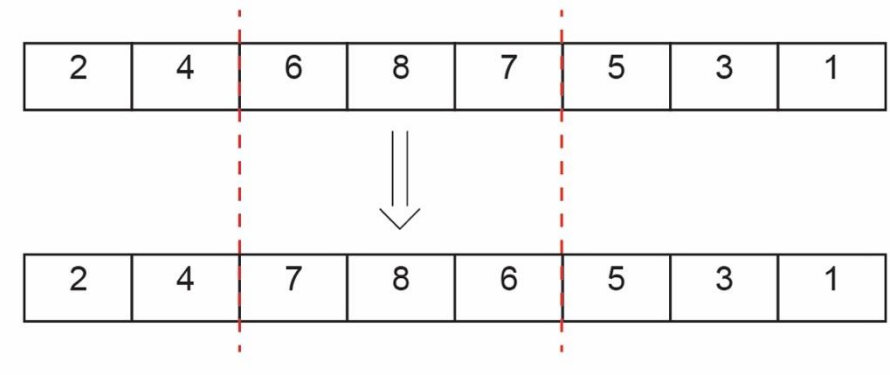


Figura 17 - Representação do operador *Inversion*

O operador ***Scramble***, representado na Figura 18, consiste em selecionar uma subsequência de genes e sortear novas posições, entre os dois pontos de corte, para cada elemento desta subsequência.

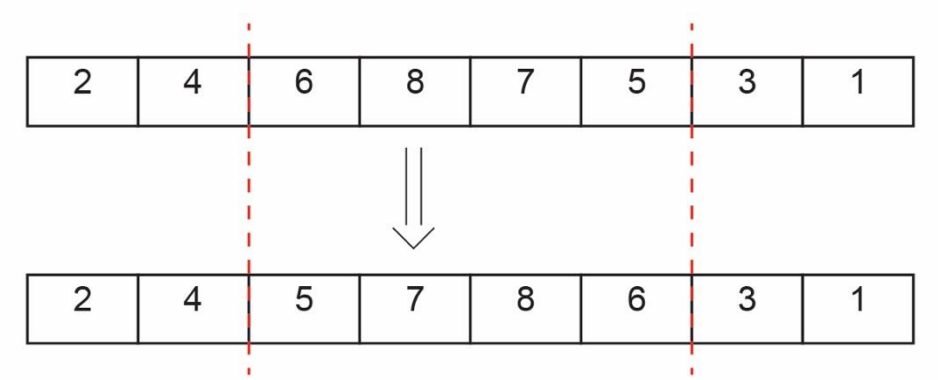


Figura 18 - Representação do operador S*cramble*

6.6.Módulo para realização de experiências

Num problema em que é usado um algoritmo genético, as experiências são particularmente importantes, pois estas permitem comparar diferentes abordagens e escolher uma em detrimento das outras. Sem a realização de experiências, seria virtualmente impossível determinar os melhores operadores para o problema que estamos a resolver.

O módulo da aplicação para realização de experiências permite definir diversos parâmetros. Um destes é o número de *runs*, que define quantas vezes o algoritmo genético é executado para cada uma das combinações dos restantes parâmetros. Como a população inicial de algoritmo genético é gerada aleatoriamente, podem-se obter *outliers* que sejam particularmente bons ou maus para um certo conjunto de parâmetros. Por isso, quantas mais *runs* forem realizados, mais populações iniciais vão ser testadas e mais fiáveis são os resultados obtidos. Para além disso, em cada *run*, a disposição dos agentes e dos *picks* é gerada aleatoriamente, evitando eventuais disposições que favoreçam certas combinações de parâmetros.

Cada *n runs* geram uma linha num ficheiro *excel*, que representa uma série de métricas relativas aos melhores indivíduos de cada um dos *n runs*. Isto é, caso sejam executados 1000 runs, as métricas são obtidas analisando os 1000 (melhores) indivíduos gerados por estes *runs*. Estas métricas são: o fitness médio e o desvio padrão do fitness médio dos indivíduos, a média e o desvio padrão do tempo dos indivíduos, a média e o desvio padrão do número de colisões por agente, o número médio de vezes que cada agente vai ao ponto de entrega, o tempo de espera médio por agente resultante de colisões, a média e mediana do tempo de espera por indivíduo para os indivíduos onde ocorrem colisões, a percentagem de *runs* em que existem colisões, a média e mediana do tempo de espera máximo para indivíduos onde existem colisões e o tempo de espera máximo de todos os *runs*.

6.7. Independência do modelo

Neste projeto foi usado o padrão de apresentação MVC, em parte, devido à independência que este confere ao modelo. Esta independência é importante devido a três grandes razões. Em primeiro lugar, dependendo do contexto, podemos querer ver a mesma informação do modelo de diferentes formas, a separação entre o modelo e a vista permite a implementação de diferentes interfaces usando o mesmo modelo. Em segundo lugar, esta independência permite a reutilização de código. E por fim, quando usamos este paradigma a manutenção de software torna-se significativamente mais fácil pois podemos alterar a vista ou o modelo sem termos que nos preocupar com o outro.

Esta independência entre o modelo e a vista é atingida usando o padrão estrutural *Observer* descrito em [18] que permite uma funcionalidade publicar/subscrever.

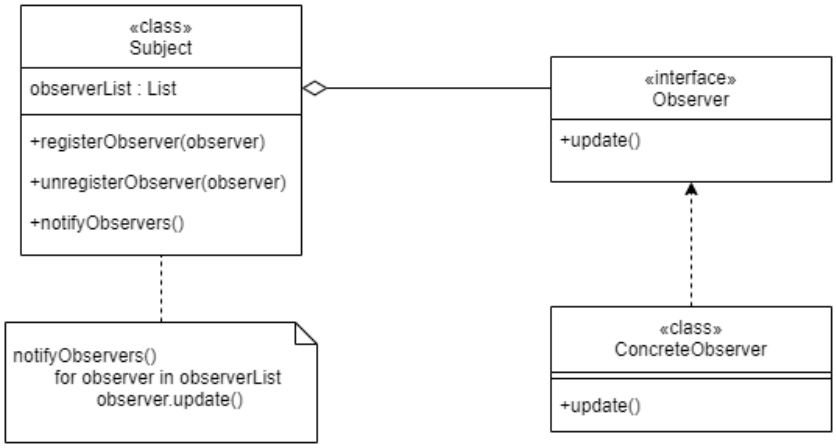


Figura 19 - Representação do padrão *Observer*

O padrão representado na Figura 19, permite que as vistas subscrevam o modelo e que sempre que este se altere estas sejam atualizadas. Em vez do modelo ter a responsabilidade de enviar informação para certas vistas, estas é que têm a responsabilidade de consumir a informação do modelo.

Neste projeto foi usado o padrão *Observer* entre a interface gráfica e o modelo, e as experiências e o modelo. Através do uso de várias técnicas de programação como interfaces, genéricos e herança conseguiu-se tornar o código o mais geral possível, o que na prática possibilita a adição de operadores ao algoritmo sem ser necessário fazer alterações adicionais ao código e facilita a reutilização deste em projetos futuros.

6.8. Testes de software

Os testes permitem demonstrar um certo nível de confiabilidade por parte da aplicação. Tendo em conta que o projeto foi desenvolvido de forma iterativa, a equipa de desenvolvimento teve a responsabilidade de detetar possíveis problemas e implementar as respetivas soluções no final de cada funcionalidade.

Tendo em conta que grande parte das funcionalidades deste projeto estão associadas ao desenvolvimento de algoritmos, recorreu-se principalmente à observação dos resultados impressos na consola e/ou na GUI para comparar com o que seria expectável. Estes testes manuais revelaram-se de extrema importância na medida em que finda cada implementação, é observado no ecrã se o resultado é o desejado permitindo validar ou não a implementação da funcionalidade. Os testes como parte integrante do processo e metodologia de desenvolvimento foram anexados a cada *sprint*. No Apêndice A apresentam-se exemplos de testes manuais realizados e que passaram com sucesso.

6.9. Descrição da interface gráfica de utilizador

Como referido anteriormente, o propósito futuro deste projeto é a integração num sistema que se encontra a ser desenvolvido e, desta forma, foi deixado claro que a interface gráfica não é a principal prioridade. Desta forma, a GUI foi implementada com o intuito de verificar visualmente os resultados das execuções dos algoritmos. Apesar disto, procurou-se desenvolver uma interface gráfica prática de usar e intuitiva.

A interface gráfica da aplicação, como é ilustrado na Figura 20, possui uma barra horizontal e uma vertical que permitem aceder a um conjunto funcionalidades. A barra de ferramentas contém botões para facilitar o acesso a funcionalidades de controlo e execução. Isto é, permite importar uma representação do armazém, importar um pedido, executar o algoritmo genético e interrompê-lo, dar início à simulação e também controlá-la com recurso a uma barra de progresso. Quanto à barra vertical permite aceder à vista do algoritmo genético, da simulação e também das experiências.

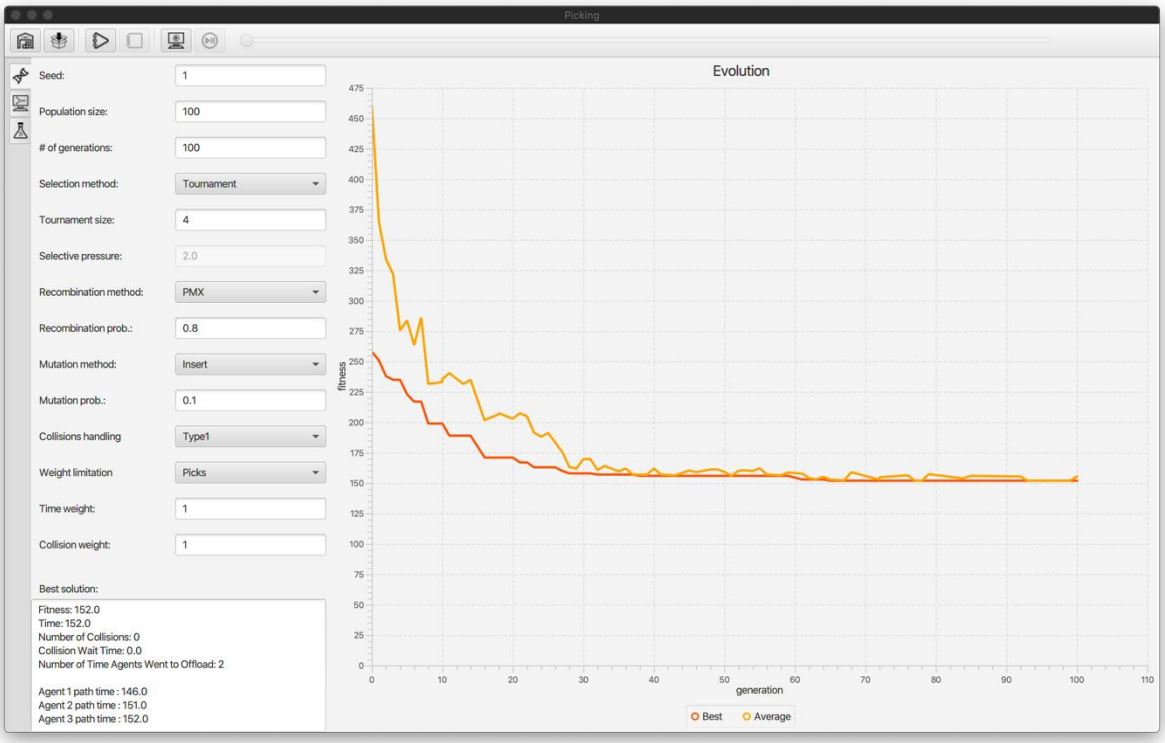


Figura 20 - Representação da interface gráfica e vista do algoritmo genético

A aplicação permite importar a partir de um ficheiro a representação de um armazém e observá-la na vista de simulação. Da mesma forma, é possível importar pedidos a partir de um ficheiro, que correspondem a produtos a serem recolhidos. Após a importação, estes produtos são representados graficamente nas prateleiras correspondentes.

Na vista do algoritmo genético é possível editar os parâmetros do algoritmo, observar a evolução ao longo das gerações através de um gráfico e verificar métricas relativas à melhor solução encontrada.

Na vista da execução de solução, como é ilustrado na Figura 21, é possível observar a representação do armazém, dos produtos nas prateleiras e as posições iniciais dos agentes. Ao executar a solução encontrada pelo algoritmo genético, o utilizador pode observar os agentes a recolherem os produtos das prateleiras, segundo a solução encontrada, finalizando com a entrega no ponto destinado para o efeito.

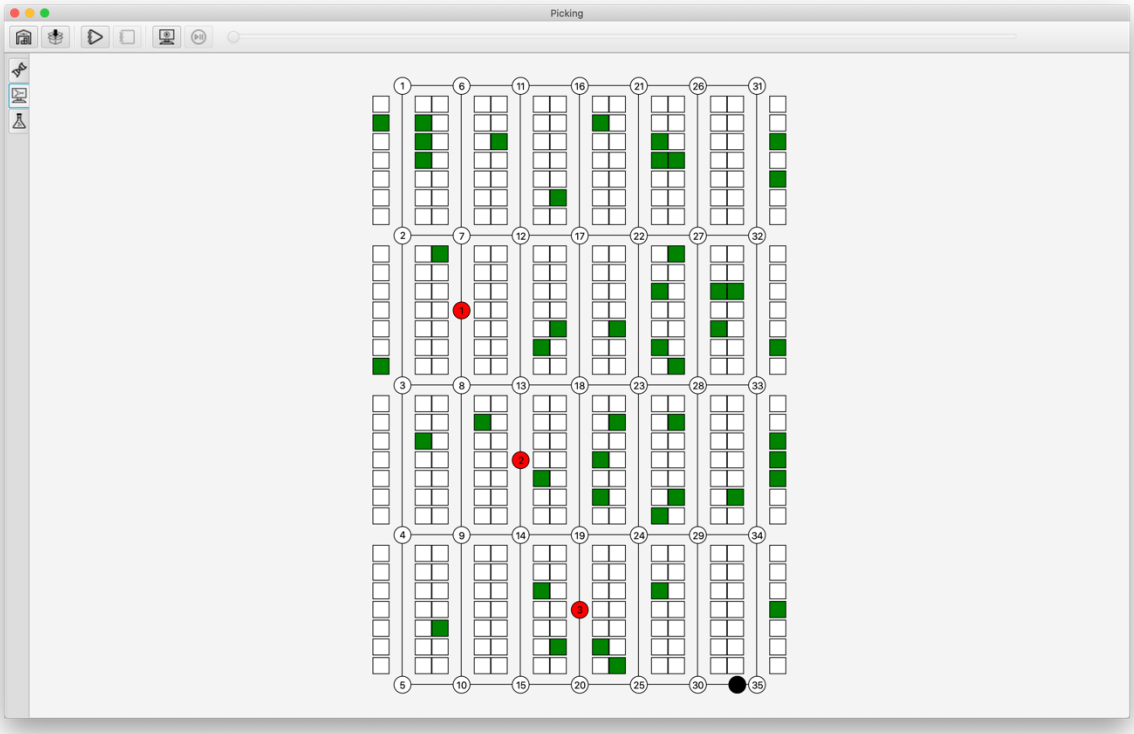


Figura 21 - Representação da vista da simulação

Numa vertente mais analítica e exploratória, como é possível observar na Figura 22, a aplicação possuiu uma funcionalidade que permite efetuar várias experiências automáticas com o objetivo de possibilitar a comparação dos parâmetros introduzidos pelo utilizador e descobrir quais os que permitem chegar à melhor solução para o problema. O utilizador apenas necessita de introduzir os parâmetros separados por vírgulas em cada caixa de texto e carregar em "*Run*". Sendo uma tarefa mais exaustiva computacionalmente o utilizador tem a oportunidade de acompanhar o progresso na barra de *status*.

A interface está preparada, tanto na vista do algoritmo genético, como na das experiências, para validar os inputs do utilizador de forma evitar possíveis exceções e assim avisar em caso de incorreta inserção de valores como é possível observar na Figura 23.

Um manual do utilizador da aplicação pode ser encontrado no Apêndice B.

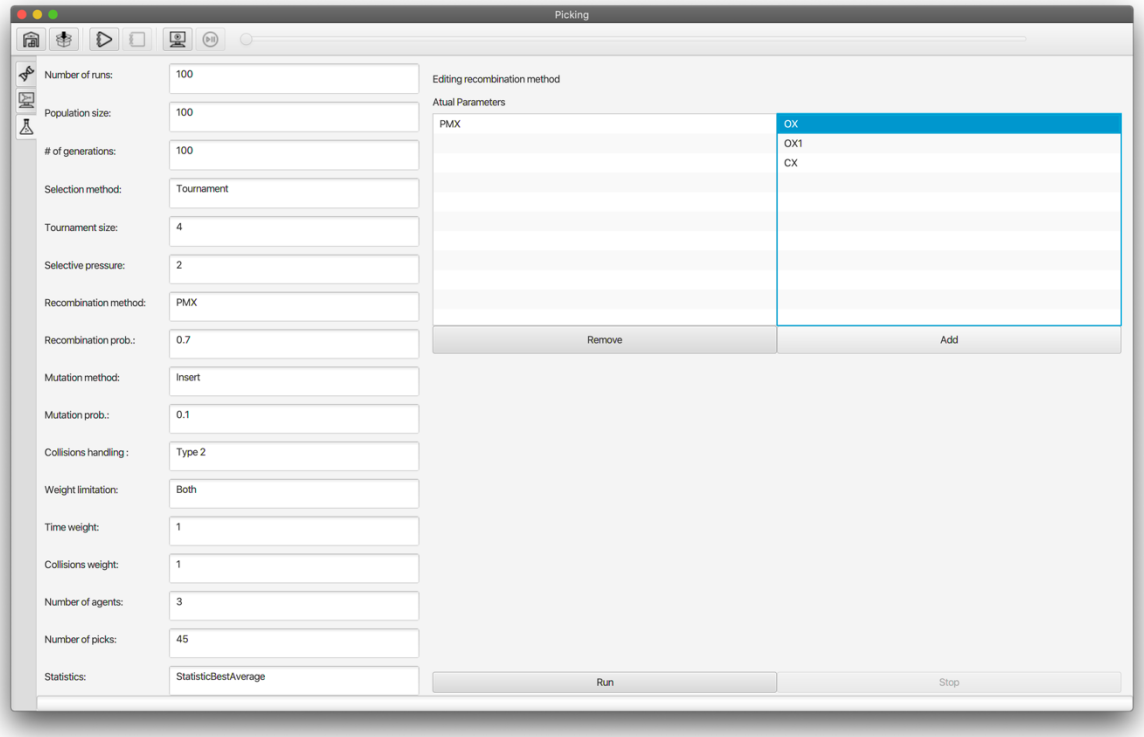


Figura 22 - Representação da vista das experiências

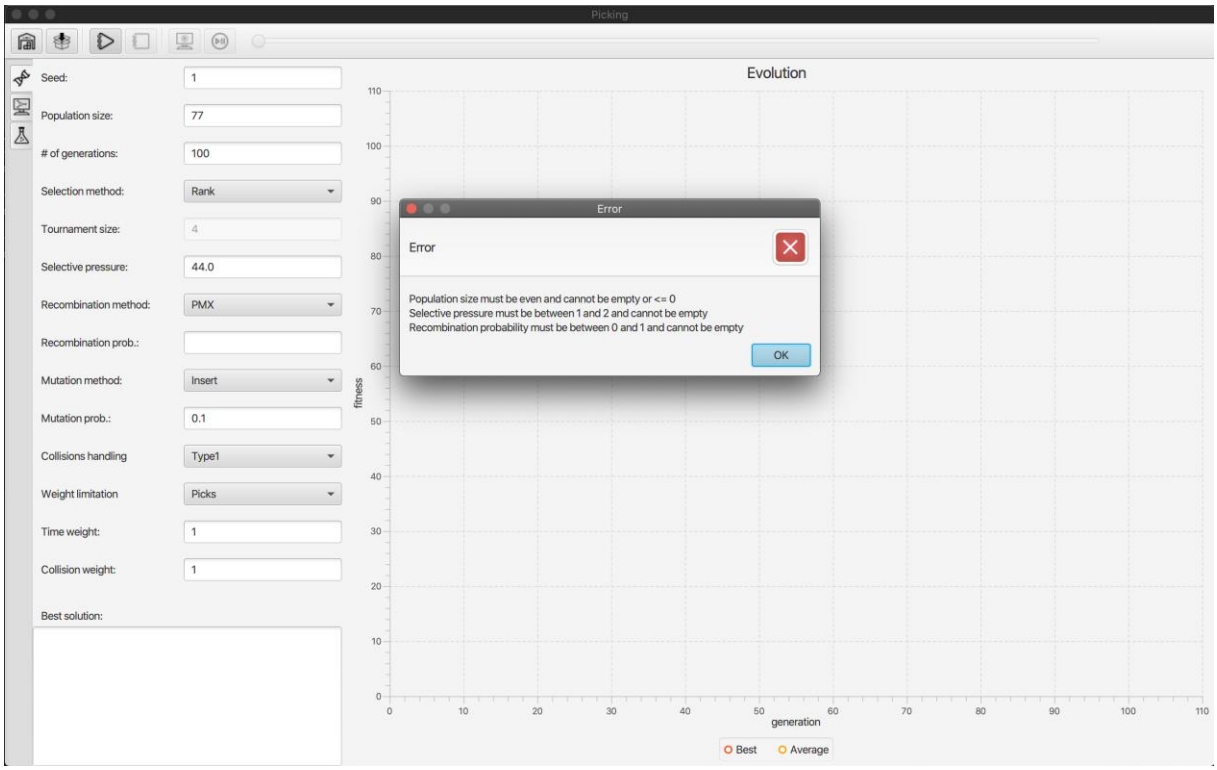


Figura 23 - Representação da janela de erros

7. Experiências e resultados

Neste capítulo são apresentados e analisados os resultados obtidos através das experiências realizadas. Com estes resultados, pretende-se determinar a viabilidade da aplicação deste projeto a uma situação (armazém) real. Pretende-se ainda, comparar as diferentes abordagens de lidar com colisões, que foram descritas no capítulo anterior (Secção 6.5.2), bem como a influência das restrições relativas ao peso que os produtos suportam e à capacidade dos agentes.

Nas experiências realizadas utilizou-se um armazém com quatro zonas horizontais, sendo que cada zona é constituída por sete corredores verticais com comprimento de sete prateleiras, como a que foi apresentada no capítulo anterior. Para comodidade do leitor apresentamos novamente a imagem da representação do armazém na Figura 24.

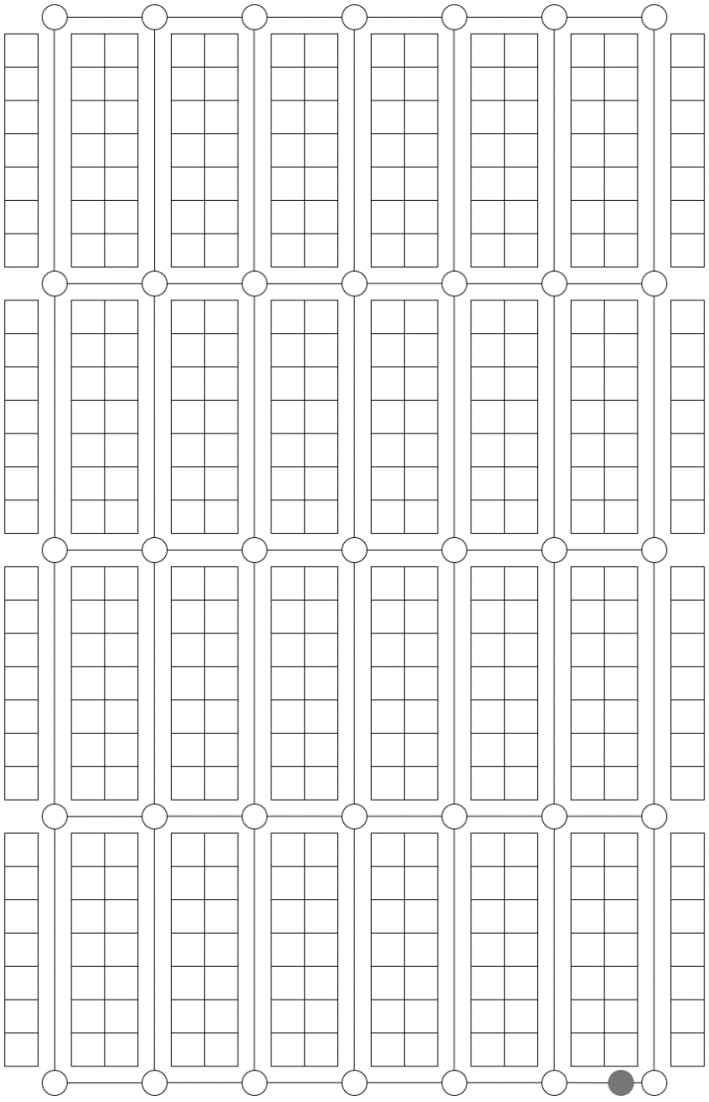


Figura 24 - Representação do armazém

Para que, os resultados obtidos sejam mais fiáveis e para que descrevam a realidade com o máximo de precisão possível, para cada combinação de parâmetros do algoritmo genético, e para cada execução é gerada aleatoriamente a posição inicial dos agentes e a posição dos *picks*. É também gerado aleatoriamente, o peso de cada produto, cujo limite inferior é 1 e o limite superior é 25, bem como o limite máximo de peso que cada produto suporta, cujo valor é gerado aleatoriamente entre 3 e 20 vezes o peso do produto. Estes limites foram impostos, com base, num inquérito informal realizado junto de um trabalhador que exerce as suas funções num armazém de distribuição. Foi ainda, definido um limite de 75, para a capacidade dos operadores que recolhem produtos das prateleiras, visto que este valor corresponde ao triplo do peso máximo dos produtos. Nas experiências realizadas usou-se $n = 1000$ para o número de *runs*.

Nas secções abaixo são apresentados resultados para as seguintes métricas: média do tempo necessário para realizar a entrega dos produtos, média do número de colisões por agente, número médio de vezes que cada agente vai ao ponto de entrega e tempo médio e máximo de espera por agente quando ocorrem colisões.

Dada a grande quantidade de parâmetros do algoritmo genético, para determinar uma combinação aceitável destes parâmetros, foi realizado um conjunto de experiências preliminares, onde foram testados diferentes tamanhos de população, operadores de seleção, recombinação e mutação tal como a pressão seletiva, tamanho do torneio e probabilidade de recombinação e mutação. Com estas experiências, determinou-se, que para este problema uma população de 100 indivíduos, 400 gerações, o operador de seleção Ordenação (*Rank*) com uma pressão seletiva de 2, o operador de recombinação PMX com uma probabilidade de recombinação de 80% e o operador de mutação *Insert* com uma probabilidade de mutação de 10% são parâmetros aceitáveis.

A primeira abordagem para lidar com colisões, usa ponderações para o tempo e para as colisões. Para determinar valores aceitáveis para estas ponderações, foi realizado um conjunto adicional de experiências. Para analisar estes resultados, recorreu-se à utilização do gráfico da Figura 25, que mostra o número de colisões médio e o tempo médio dos indivíduos, para cada combinação destas ponderações. Neste gráfico, um ponto (correspondente a uma combinação de ponderações) é objetivamente melhor do que todos os pontos que tenham simultaneamente valor mais elevado de tempo e número de colisões. Diz-se que um ponto é não dominado se não for objetivamente pior do que nenhum outro ponto (pontos a verde no gráfico). Após a análise deste gráfico, selecionaram-se as combinações de ponderações (8, 6) e (4, 10), por não

serem dominadas por outras combinações e por representarem um compromisso entre as duas dimensões (tempo e número de colisões).

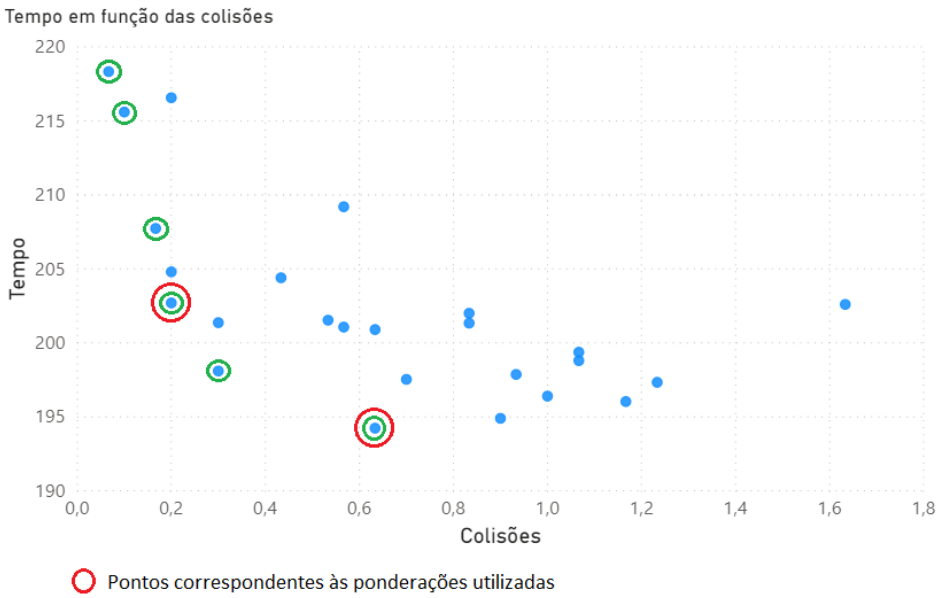


Figura 25 - Representação do tempo e das colisões para diferentes ponderações

Foi ainda determinado, através da visualização de diversas execuções do algoritmo, que 400 gerações são suficientes para a população convergir, sendo este o valor que foi utilizado nas experiências realizadas.

Nas experiências cujos resultados se apresentam abaixo, são comparadas as diferentes abordagens de lidar com as colisões. Na primeira abordagem, são atribuídas ponderações ao tempo e ao número de colisões de cada indivíduo. A segunda abordagem, consiste em determinar o tempo, que seria necessário que cada agente esperasse no ponto de decisão, para resolver cada colisão e adicionar à *fitness* do indivíduo, o menor destes tempos. Por fim, a terceira abordagem, consiste também em determinar o tempo que cada indivíduo tem de esperar no ponto de decisão para resolver a colisão, mas atribui-se à *fitness* o tempo de espera do agente que, resolvida a colisão, apresente um menor tempo para chegar ao destino. Apresentam-se e analisam-se também resultados para experiências em que foram consideradas as restrições relativas tanto à capacidade dos agentes como às restrições de peso dos produtos, mas também experiências em que apenas foi considerada a capacidade dos agentes. Estas últimas experiências justificam-se por existirem situações em que não existem restrições significativas relativamente ao peso que os produtos suportam.

7.1. Abordagem baseada em ponderações

Na Figura 26, é apresentado o tempo médio para a recolha de todos os produtos de uma encomenda, para as ponderações (8, 6) e (4, 10) quando se consideram para as restrições de capacidade dos agentes e produtos. Na Figura 27 é apresentada a mesma métrica, mas apenas tendo em conta a capacidade dos agentes.

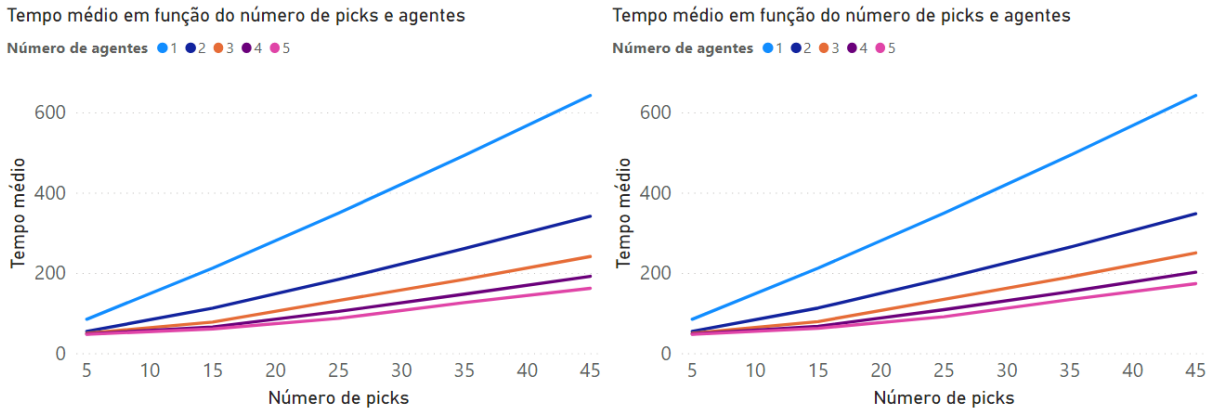


Figura 26 - Representação do tempo em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para as combinações de ponderações (8, 6) (esquerda) e (4, 10) (direita)

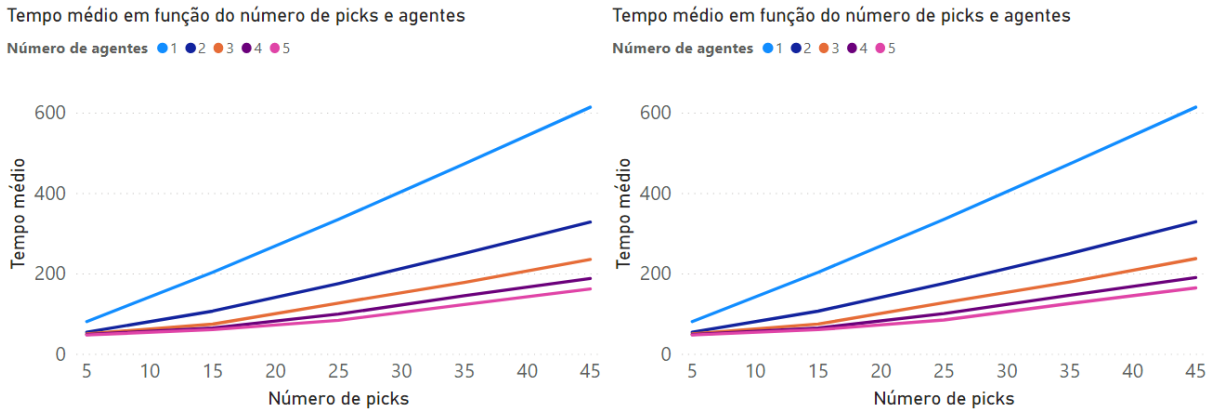


Figura 27 - Representação do tempo em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para as combinações de ponderações (8, 6) (esquerda) e (4, 10) (direita)

Os resultados obtidos usando as ponderações (8, 6) apresentam um menor tempo médio relativamente às ponderações (4, 10), o que está de acordo com os resultados esperados, visto que o tempo tem uma ponderação maior no primeiro caso, ou seja, tem mais importância do que o número de colisões na avaliação do indivíduo. Pode ainda ser observado que quando apenas se impõe a restrição de capacidade dos agentes o tempo médio é menor, resultado

também esperado uma vez que os agentes nunca interrompem a recolha de produtos por se ter excedido a capacidade dos produtos. Nestes gráficos também se pode observar que o tempo médio de entrega dos produtos diminui com o número de agentes e aumenta com o número de *picks* de forma aproximadamente linear. Este resultado verifica-se em todas as experiências realizadas.

Na Figura 28, é apresentado o número médio de colisões, que ocorrem durante o processo de recolha dos produtos das prateleiras, para as ponderações (8, 6) e (4, 10) quando se tem em conta as restrições de capacidade dos agentes e produtos em conjunto, enquanto que na Figura 29 é apresentada a mesma métrica, mas apenas tendo em conta a capacidade dos agentes.

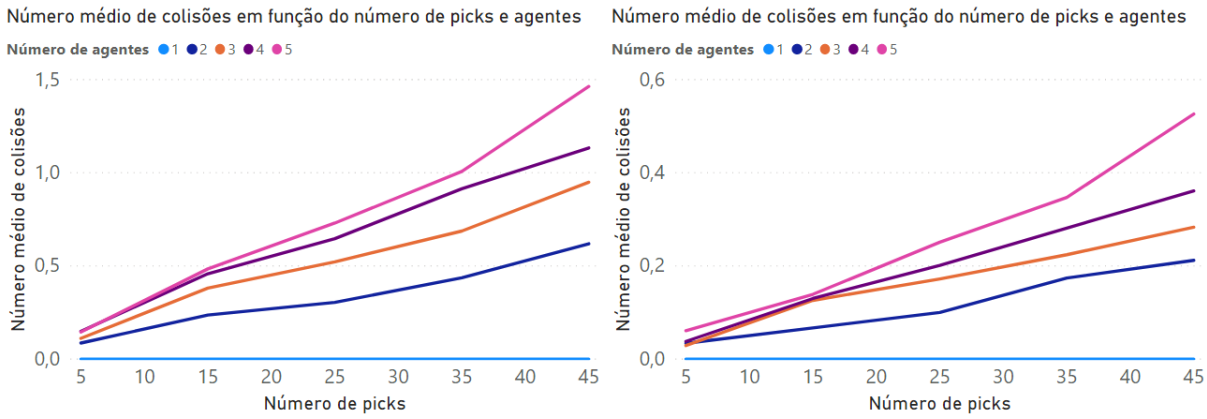


Figura 28 - Representação do número de colisões em função do número de *picks* e agentes tendo em conta a capacidade dos agentes e produtos para as ponderações (8, 6) (esquerda) e (4, 10) (direita)

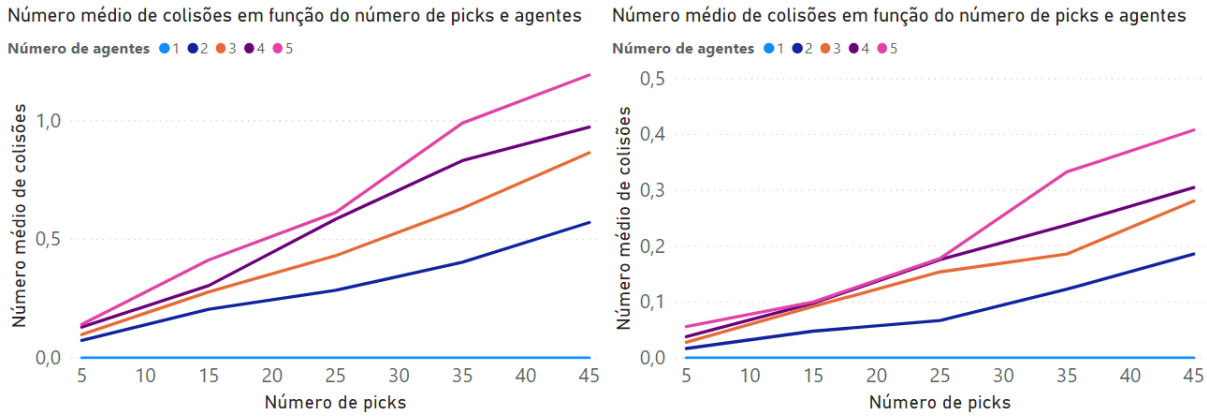


Figura 29 - Representação do número de colisões em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para as ponderações (8, 6) (esquerda) e (4, 10) (direita)

Como pode ser observado, os indivíduos gerados, usando as ponderações (4, 10) apresentam, em média, um número menor de colisões, o que vai de encontro ao esperado, visto que é atribuída uma ponderação maior às colisões. Quando é considerada apenas a capacidade dos

agentes, o número de colisões é inferior. Este resultado pode ser explicado pelo facto de, se bem sucedido na distribuição dos *picks* pelos agentes, o algoritmo vai tender a atribuir *picks* aos agentes em diferentes zonas do armazém, o que leva indiretamente a uma diminuição do número de colisões. Quanto maior o número de vezes que os agentes têm que interromper a recolha dos produtos para ir ao ponto de entrega (o que ocorre mais vezes se se tiver em conta a restrição da capacidade dos produtos), maior será a probabilidade de ocorrência de colisões, uma vez que os agentes tenderão a passar mais vezes em zonas frequentadas por outros agentes.

Pode ainda ser observado no gráfico que, naturalmente, não ocorrem colisões quando se utiliza apenas um agente.

Na Figura 30, é apresentado o número de deslocações ao ponto de entrega, que cada agente realiza devido ao excesso da capacidade do próprio agente ou produtos para as ponderações (8, 6) e (4, 10) e na Figura 31 é apresentada a mesma métrica, mas apenas tendo em conta a capacidade dos agentes.

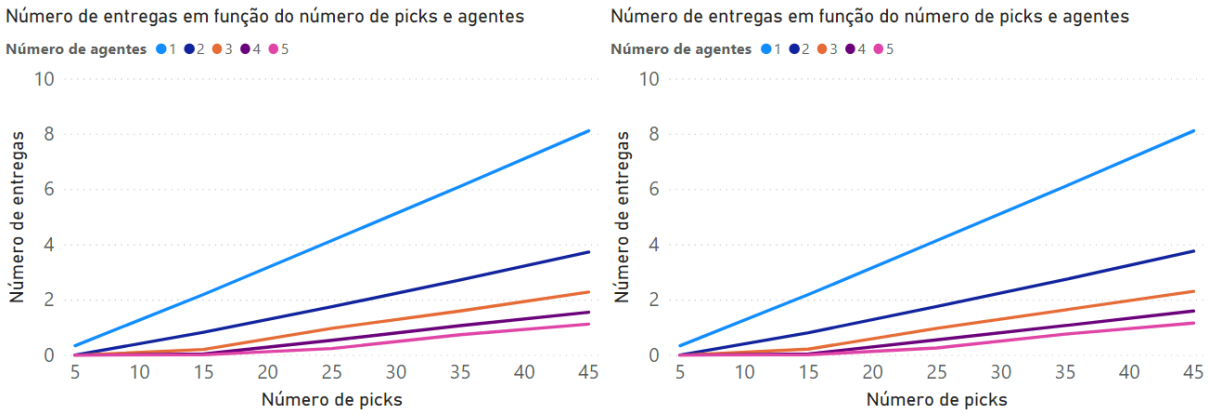


Figura 30 - Representação do número de deslocações ao ponto de entrega por cada agente em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para as ponderações (8, 6) (esquerda) e (4, 10) (direita)

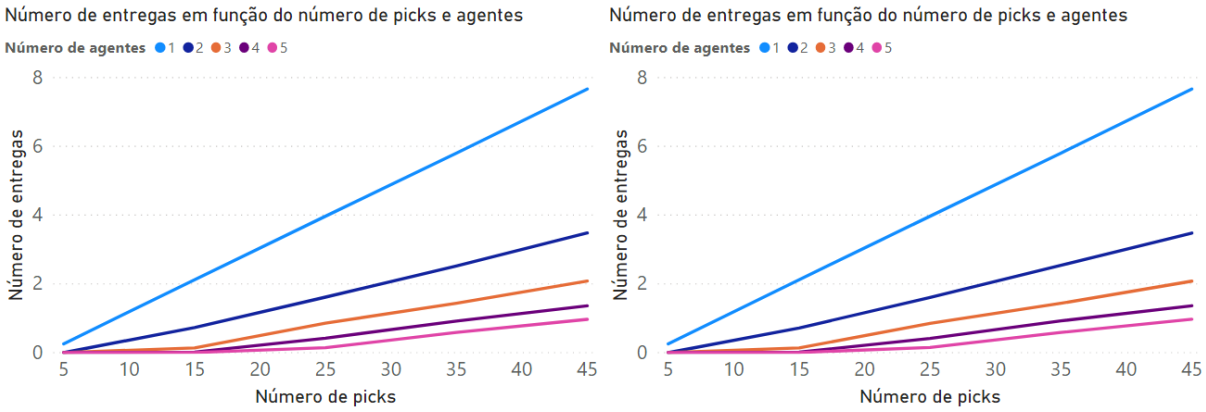


Figura 31 - Representação do número de deslocações ao ponto de entrega por cada agente em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para as ponderações (8, 6) (esquerda) e (4, 10) (direita)

Como demonstram os resultados, os indivíduos gerados com as ponderações (8, 6) apresentam em média um menor número de deslocações ao ponto de entrega, o que coincide com o esperado, visto que, o número de entregas afeta, apesar de indiretamente, o parâmetro tempo, que usando estas ponderações tem uma maior importância. É ainda observável que quando se aplica apenas a restrição de capacidade dos agentes, o número de entregas é ligeiramente menor.

7.2. Abordagens que penalizam o tempo de espera para evitar colisões

Na Figura 32 é apresentado o tempo médio de recolha, tendo em conta as restrições de capacidade dos agentes e produtos e na Figura 33 é apresentada a mesma métrica, mas apenas tendo em conta a capacidade dos agentes.

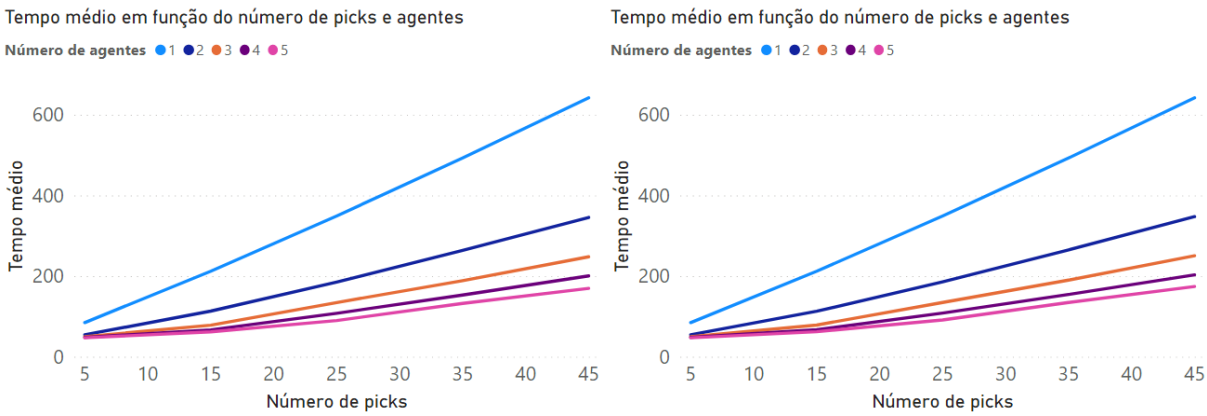


Figura 32 - Representação do tempo em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens

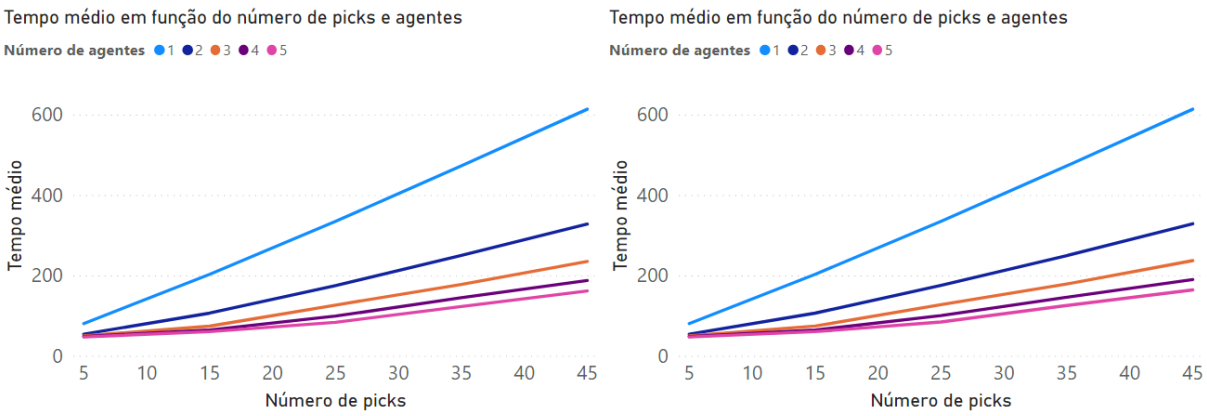


Figura 33 - Representação do tempo em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para a segunda (esquerda) e terceira (direita) abordagens

Os resultados obtidos, mostram que a segunda abordagem, apresenta uma redução no tempo de recolha quando comparada com a terceira abordagem. Pode ainda observar-se que o tempo é menor quando se tem em conta apenas a restrição de capacidade dos agentes, tal como esperado.

Na Figura 34, é apresentado o número médio de colisões, que ocorrem durante o processo de recolha dos produtos tendo em conta as restrições de capacidade para os agentes e produtos. Na Figura 35 apenas se considera a capacidade dos agentes.

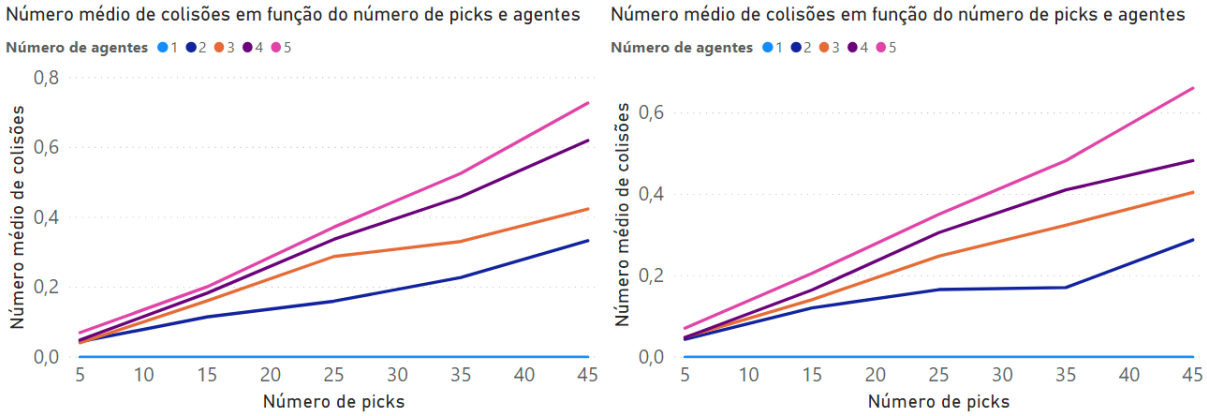


Figura 34 - Representação do número de colisões em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens

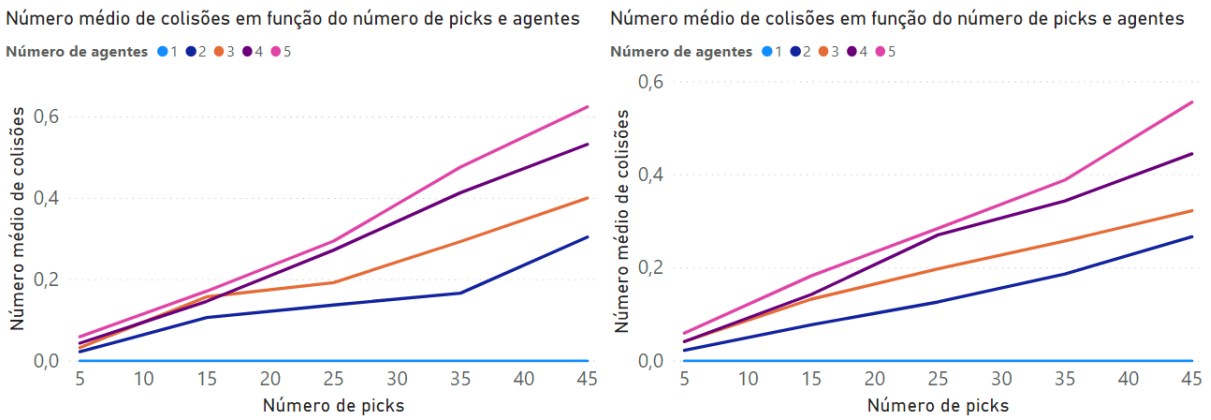


Figura 35 - Representação do número de colisões em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para a segunda (esquerda) e terceira (direita) abordagens

Como pode ser observado, ambas as abordagens apresentam um número de colisões muito baixo, sendo que a terceira abordagem, quando comparada com a segunda, apresenta uma pequena diminuição no número de colisões. Pode ainda observar-se que o número médio de colisões é menor quando se tem conta apenas a restrição de capacidade dos agentes, tal como se verifica quando se utilizam ponderações.

Na Figura 36, é apresentado o número de deslocações ao ponto de entrega, que cada agente realiza devido ao excesso da capacidade do próprio agente ou produtos para a segunda e terceira abordagens. Na Figura 37 é apresentada a mesma métrica, mas apenas tendo em conta a capacidade dos agentes.

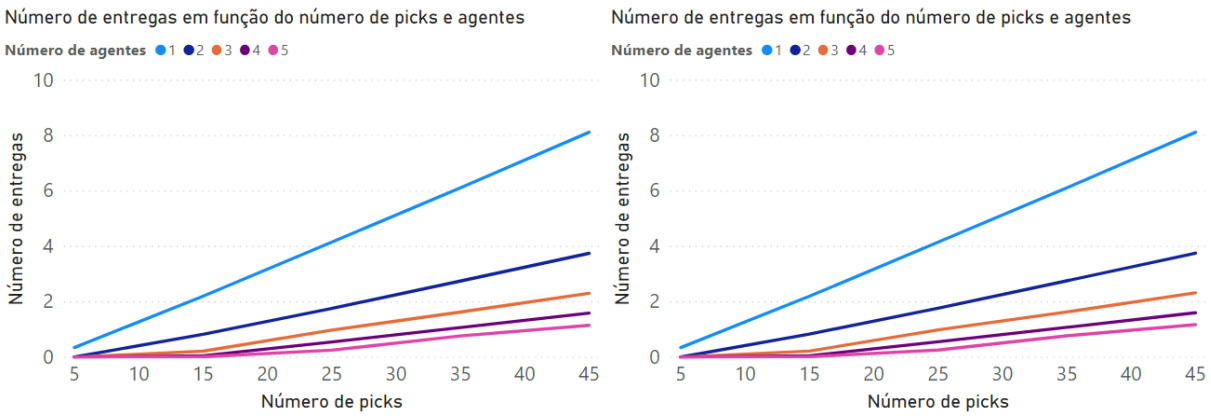


Figura 36 - Representação do número de deslocações ao ponto de entrega por cada agente em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens

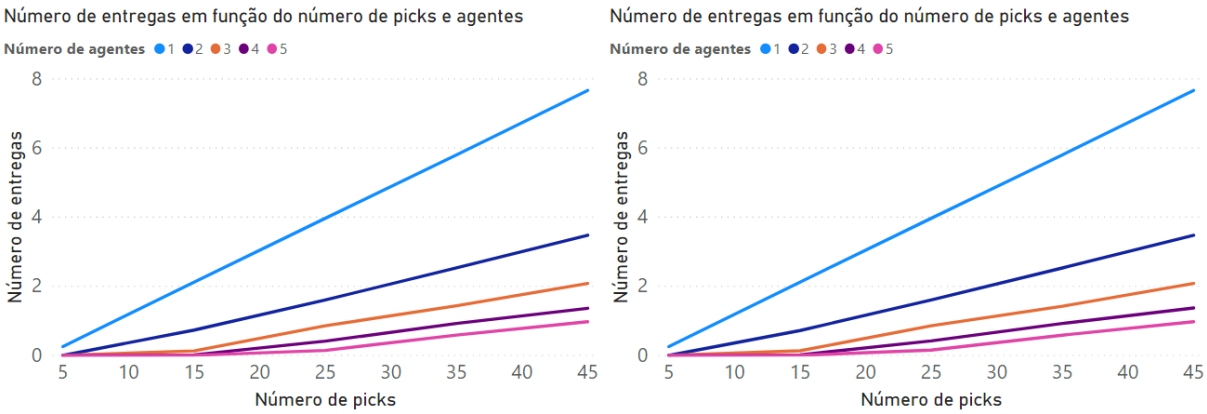


Figura 37 - Representação do número de deslocações ao ponto de entrega por cada agente em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens

Nestes gráficos podemos observar que o número de deslocações ao ponto de entrega é muito semelhante para as duas abordagens. Para além disso, é possível observar um menor número de deslocações ao ponto de entrega quando considerada apenas a capacidade dos agentes.

Na Figura 38, é apresentada média da soma dos tempos de espera de todas as colisões que resultam do processo de *picking* tendo em conta as restrições de capacidade para os agentes e produtos. Na Figura 39 é considerada apenas a restrição de capacidade dos agentes.

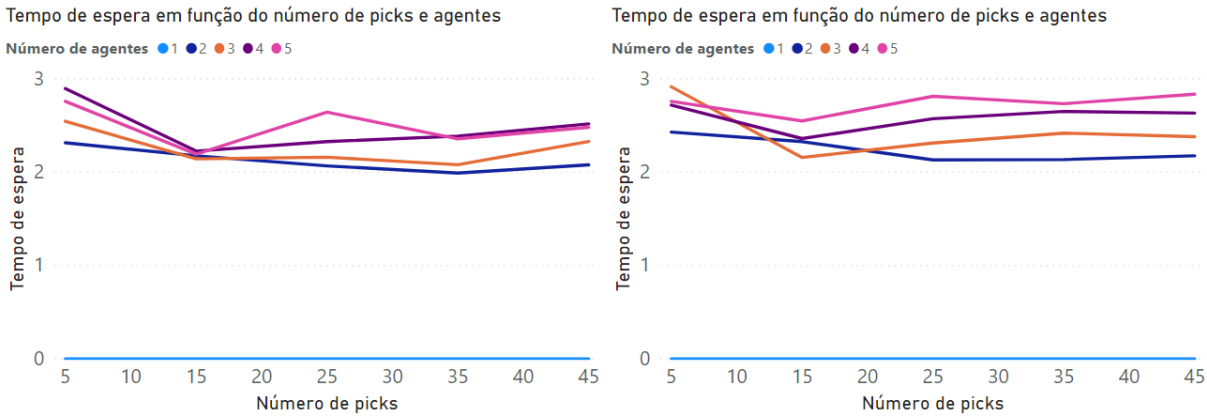


Figura 38 - Representação da média da soma dos tempos de espera em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens

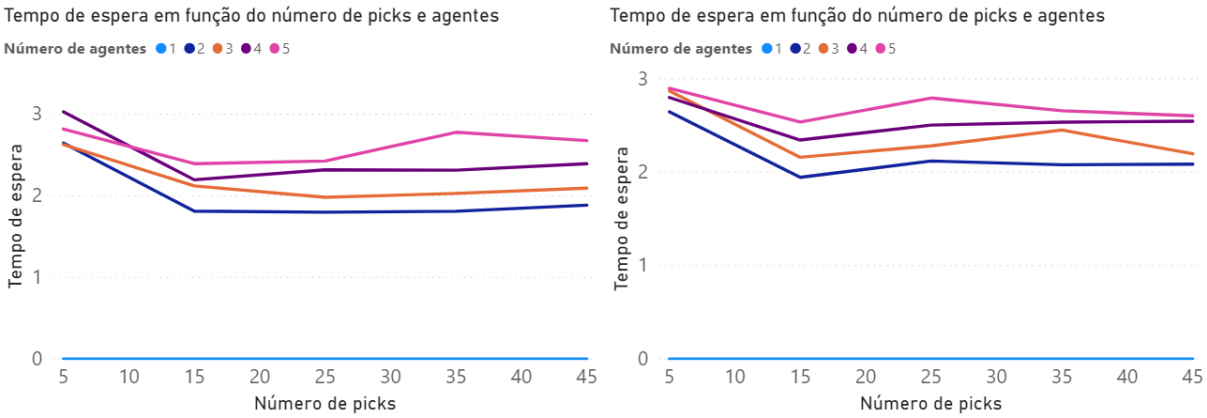


Figura 39 - Representação da média da soma dos tempos de espera em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para a segunda (esquerda) e terceira (direita) abordagens

Analisando os resultados, chega-se à conclusão de que, sempre que existem colisões, o tempo de espera, para ambas as abordagens, é muito baixo (sempre inferior a 3), sendo que, para a segunda abordagem são ligeiramente mais baixos. Estes resultados, permitem assim concluir que, quando existe a possibilidade de ocorrência de colisões, o tempo que os agentes têm que esperar em pontos de decisão para que estas não ocorram é baixo, em média. Quando considerada a capacidade dos agentes é observável uma ligeira diminuição do tempo médio de espera.

Na Figura 40 é apresentada a média do tempo de espera máximo, ou seja, o tempo da colisão que demorou mais tempo a resolver, de entre todas as colisões que ocorreram para cada indivíduo.

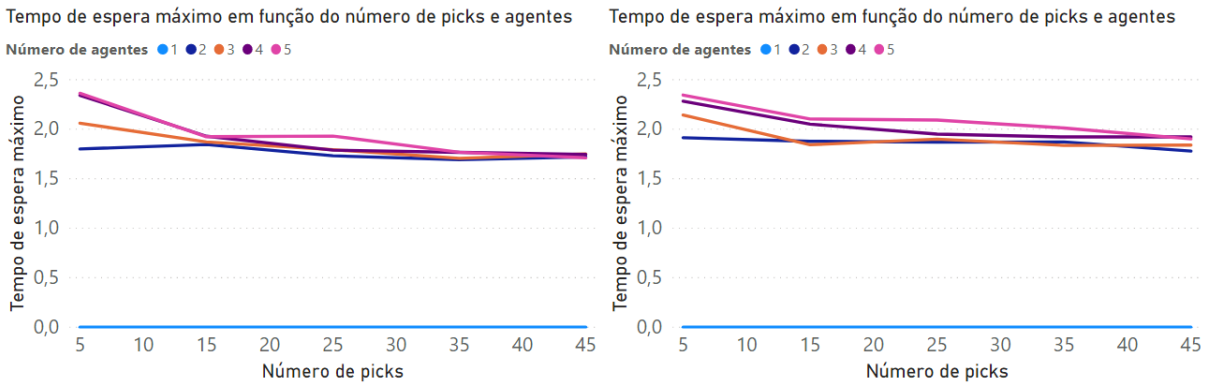


Figura 40 - Representação do tempo de espera máximo em função do número de *picks* e agentes em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes e produtos para a segunda (esquerda) e terceira (direita) abordagens

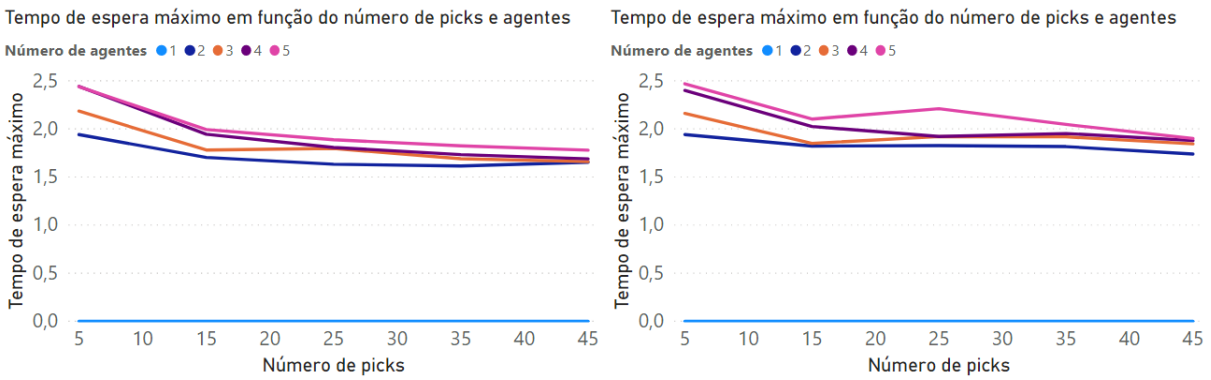


Figura 41 - Representação do tempo de espera máximo em função do número de *picks* e agentes em função do número de *picks* e agentes, tendo em conta a capacidade dos agentes para a segunda (esquerda) e terceira (direita) abordagens

Estes resultados, mostram que todas as colisões que ocorrem, são resolvidas rapidamente, não gerando assim grande transtorno no processo de *picking*. Podemos ainda observar que a segunda abordagem apresenta ligeiramente melhores resultados. É ainda de referir, que para o armazém estudado, o limite máximo de tempo para resolver uma colisão usando a segunda abordagem é cinco, enquanto que com a terceira abordagem é oito. Isto acontece, pois, os corredores do armazém têm um comprimento de sete unidades, e a segunda abordagem escolhe que o agente que espera, é sempre aquele que apresenta um menor tempo para resolver a colisão, enquanto a terceira abordagem pode escolher o que apresenta o maior tempo. Ou seja, e de acordo com o que foi descrito na Secção 6.5.2, relativo à segunda abordagem, o agente que espera tem uma distância máxima para o ponto de colisão igual a metade do comprimento do corredor e na terceira abordagem o agente que espera pode ter que percorrer todo o corredor.

7.3. Discussão de resultados

A abordagem baseada em ponderações, permite que o utilizador defina o parâmetro ao qual quer dar mais importância (atribuindo uma ponderação maior), enquanto as abordagens que penalizam o tempo de espera, tratam deste equilíbrio entre colisões e tempo para realizar a recolha de todos os produtos de uma forma orgânica, fazendo assim com que o utilizador não tenha de procurar um conjunto de ponderações aceitável.

Com estas experiências, comparando a abordagem baseada em ponderações com as abordagens que penalizam o tempo de espera, conclui-se que a primeira apresenta um aumento no número de colisões quando o tempo diminui e vice-versa, enquanto a outras encontram um melhor compromisso entre estes dois parâmetros. Para além disso, as abordagens que penalizam o

tempo de espera têm em consideração a severidade destas colisões, tentando assim minimizar o tempo de espera, o que não acontece na primeira abordagem.

Quando se comparam a segunda e terceira abordagens, as quais penalizam o tempo de espera, os resultados mostram que estas apresentam métricas muito semelhantes. No entanto, podemos concluir que a segunda abordagem é ligeiramente melhor, visto que apresenta um valor mais baixo para o tempo de recolha de todos os produtos, para o tempo de espera, para a resolução de colisões e para o tempo de espera máximo, sendo que o número de colisões é semelhante em ambas abordagens. É possível ainda concluir, que quando apenas é tida em consideração a capacidade dos agentes, obtêm-se resultados ligeiramente melhores.

Em todas as experiências efetuadas, verificou-se que o tempo médio de recolha dos produtos e o número de deslocações ao ponto de entrega, diminui com o número de agentes e aumenta com o número de *picks* de forma aproximadamente linear. Pode ainda observar-se que o número de colisões aumenta com o número de agentes e com o número de *picks*. Pode concluir-se assim, que a solução apresentada é boa tendo em conta o baixo número de colisões para todos os casos testados.

8. Conclusão

O processo de *picking* em armazéns, é uma tarefa essencial e por vezes muito dispendiosa. Neste projeto foi desenvolvida uma aplicação que permite otimizar o processo de recolha de produtos num armazém através da distribuição destes produtos e a ordem pela qual estes são recolhidos. Tendo como propósito a otimização deste processo, recorreu-se a uma meta-heurística e a um algoritmo de procura, algoritmo genético e A*, respetivamente. Restrições como a capacidade dos produtos, agentes ou ambos e várias abordagens para lidar com colisões foram tidas em conta e implementadas soluções. Minimizar o número de colisões durante o processo de recolha de produtos, é um aspeto importante na resolução deste problema, visto isto, considera-se que o algoritmo implementado apresenta uma boa capacidade para resolver este problema. Entre as diferentes abordagens implementadas, observou-se que a abordagem que penaliza o tempo de espera do agente que tem que esperar menos tempo para evitar a colisão, é melhor para lidar com colisões visto que apresenta um valor mais baixo para o tempo de recolha de todos os produtos, para o tempo de espera, para a resolução de colisões e para o tempo de espera máximo.

Verificou-se ainda que o tempo médio de recolha dos produtos e o número de deslocações ao ponto de entrega, diminui com o número de agentes e aumenta com o número de *picks* de forma aproximadamente linear. Pode ainda observar-se que o número de colisões aumenta com o número de agentes e com o número de *picks*.

Os objetivos deste projeto foram cumpridos com sucesso, tendo sido desenvolvidas diferentes abordagens para tratar colisões entre os operadores, bem como lidar com as restrições de peso, capacidade de cada produto e operadores.

Quanto a trabalho futuro, destaca-se a integração dos algoritmos desenvolvidos com os restantes sistemas do projeto global em que este projeto estava enquadrado, e possíveis adaptações necessárias para realizar essa mesma integração.

Referências Bibliográficas

[1] Dallari, F., Marchet, G. & Melacini, M. (2009). Design of order picking system. Int J Adv Manuf Technol 42, 1–12. https://doi.org/10.1007/s00170-008-1571-9.

[2] André Scholz, Sebastian Henn, Meike Stuhlmann , Gerhard Wäscher (2016). A new mathematical programming formulation for the Single-Picker Routing Problem, European Journal of Operational Research, 253 (1), 68-84.

[3] Z. Zhang, Q. Guo and P. Yuan (2017), Conflict-free route planning of automated guided vehicles based on conflict classification, *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Banff, AB, 5-8 Oct. 2017, pp. 1459-1464, DOI: 10.1109/SMC.2017.8122819

[4] Pavel Surynek (2019), Conflict Handling Framework in Generalized Multi-agent Path Finding: Advantages and Shortcomings of Satisfiability Modulo Approach ,11th International Conference on Agents and Artificial Intelligence, Prague, Czech Republic, 19-21 Feb. 2019, pp. 192-203, DOI: 10.5220/0007374201920203

[5] Thomas Chabot, Rahma Lahyani, Leandro C. Coelho & Jacques Renaud (2017), Order picking problems underweight, fragility and category constraints, *International Journal of Production Research*, 55 (21), 6361-6379, DOI: 10.1080/00207543.2016.1251625

[6] Pansart, L., Nicolas, C. & Cambazard, H. (in press). Exact algorithms for the picking problem. Elsevier. Disponível em https://hal.archives-ouvertes.fr/hal-01481915v3.

[7] Roodbergen, K. J. & Koster, R. (2001). Routing methods for warehouses with multiple cross aisles. *International Journal of Production Research, 39*(9), 1865-1883, DOI: 10.1080/00207540110028128.

[8] Daniels, R. L., Rummel, J. L. & Schantz, R. (1998). A model for warehouse order picking. *European Journal of Operational Research, 105*, 1-17.

[9] Henn, S., Koch, S., Doerner, K. F., Strauss, C. & Wäschera, G. (2010). Metaheuristics for the Order Batching Problem in Manual Order Picking Systems. *Verband der Hochschullehrer für Betriebswirtschaft e.V., 3*(1), 82-105.

[10] Hsu, C.-M., Chen, K.-Y. & Chen, M.-C. (2005). Batching orders in warehouses by minimizing travel distance with genetic algorithms, *Computers in Industry*, 56(2), 169–178.

[11] Chatting, M., Craven, M. (2018). A Comparison of Exact and Heuristic Algorithms to Solve the Travelling Salesman Problem. *The Plymouth Student Scientist, 11*(2), 53-91.

[12] Damghanijazi, E. & Mazidi, A. (2017). Meta-Heuristic Approaches for Solving Travelling Salesman Problem. *International Journal of Advanced Research in Computer Science, 8*(5), 18-23.

[13] Ernesto Costa, Anabela Simões (2011), Inteligência Artificial – Fundamentos e Aplicações (3ª Edição). Lisboa: FCA – Editora de Informática, Lda.

[14] Ken Schwaber and Jeff Sutherland (2017), The Scrum Guide. Consultado em 10 de março de 2020. Disponível em https://www.scrumguides.org/scrum-guide.html

[15] D. J Anderson and A. Carmichael, Essential Kanban Condensed | Lean Kanban. Consultado em 10 de março de 2020. Disponível em https://leankanban.com/guide

[16] Don Wells (2009). Extreme programming. Consultado em 10 de março de 2020. Disponível em http://www.extremeprogramming.org/

[17] Fowler, M. (2003). Patterns of enterprise application architecture. Boston: Addison-Wesley.

[18] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. M. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

[19] EDUCBA - Corporate Bridge Consultancy Pvt Ltd. Java Swing vs Java FX. Consultado em 20 junho 2020. Disponível em https://www.educba.com/java-swing-vs-java-fx/

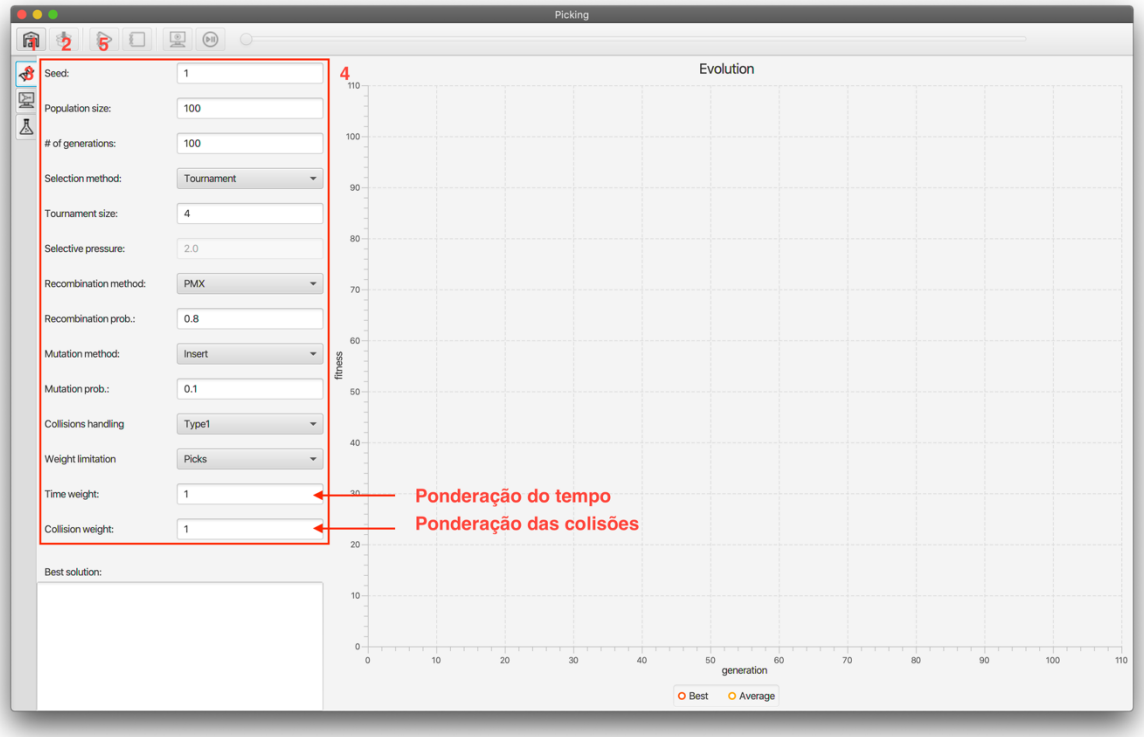
Apêndice A – Testes manuais realizados

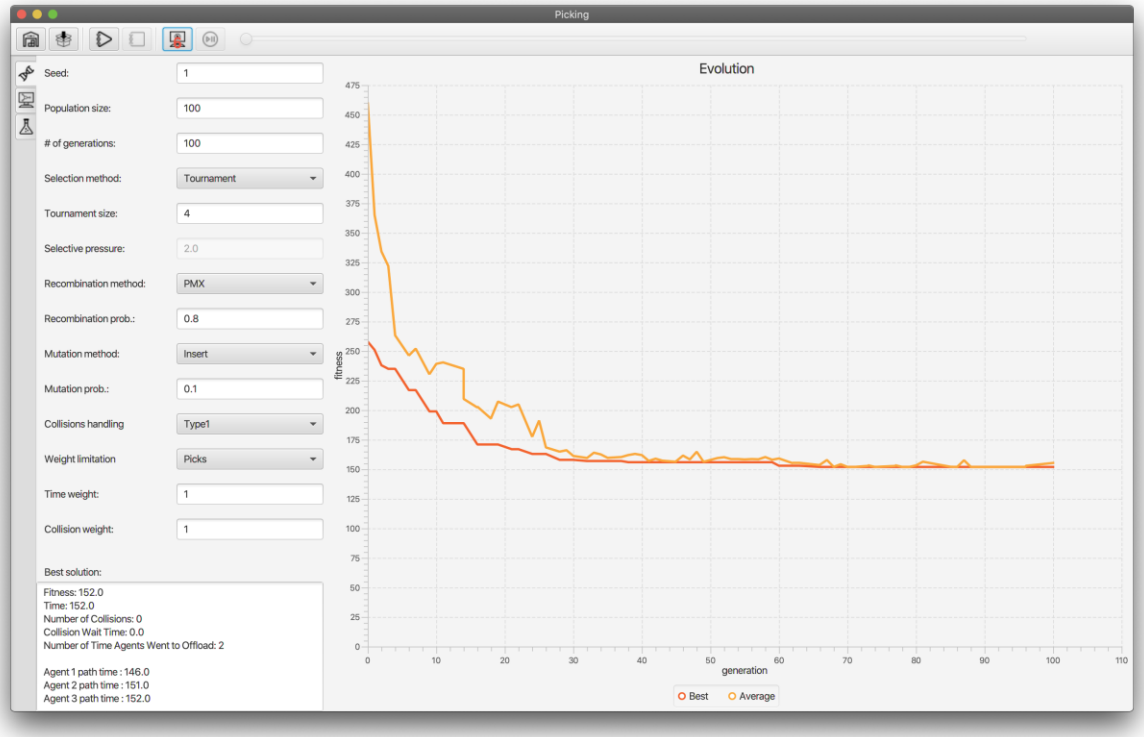
- Comparação com o output expectável dos descendentes produzidos pelos operadores de seleção, recombinação e mutação através de prints na console.
- Observação do indivíduo e verificação se a sua *fitness* corresponde à fitness calculada.
- Comparação do melhor indivíduo resultante do algoritmo genético com a simulação.
- Observação na GUI se a representação do armazém era a esperada.
- Comparação do número de colisões calculado com o expectado.
- Testar GUI com diferentes armazéns.
- Verificar que o caminho entre um par de nós calculado pelo A* não é calculado mais do que uma vez.
- Verificar que o número de colisões de cada indivíduo é o esperado.
- Verificar que o peso calculado corresponde à realidade.
- Imprimir cada indivíduo e verificar manualmente que este vai ao ponto de descarga apenas quando já não pode recolher mais produtos tendo em consideração a capacidade de cada produto.
- Verificar se a penalização adicionada a cada agente corresponde ao esperado.
- Verificar se a distância calculada do ponto de colisão ao nó de decisão mais próximo corresponde ao esperado.

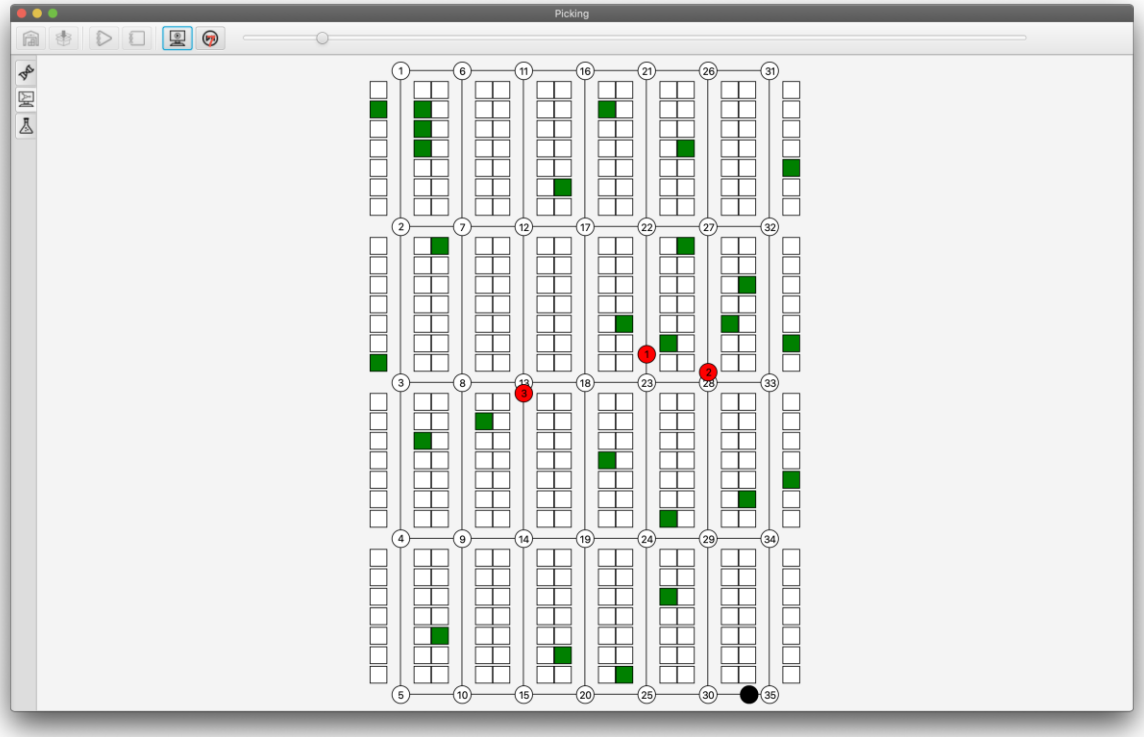
Apêndice B – Manual de utilizador

Correr algoritmo genético e visualizar a simulação da solução

1. Importar o armazém a partir do ficheiro (WarehouseLayout.json);
2. Importar os *picks* a partir do ficheiro (Picks.json);
3. Mudar para a vista do algoritmo genético;
4. Editar parâmetros;
5. Executar algoritmo genético;
6. Premir no botão para executar a simulação;
7. Play/Pause.

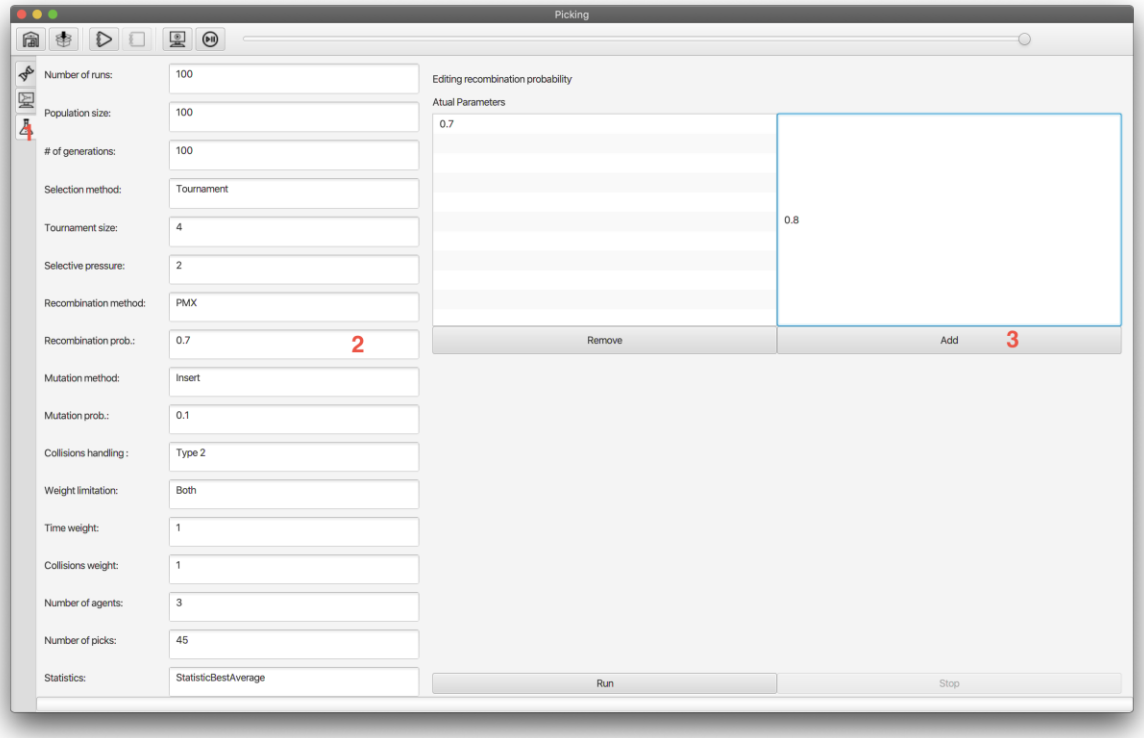


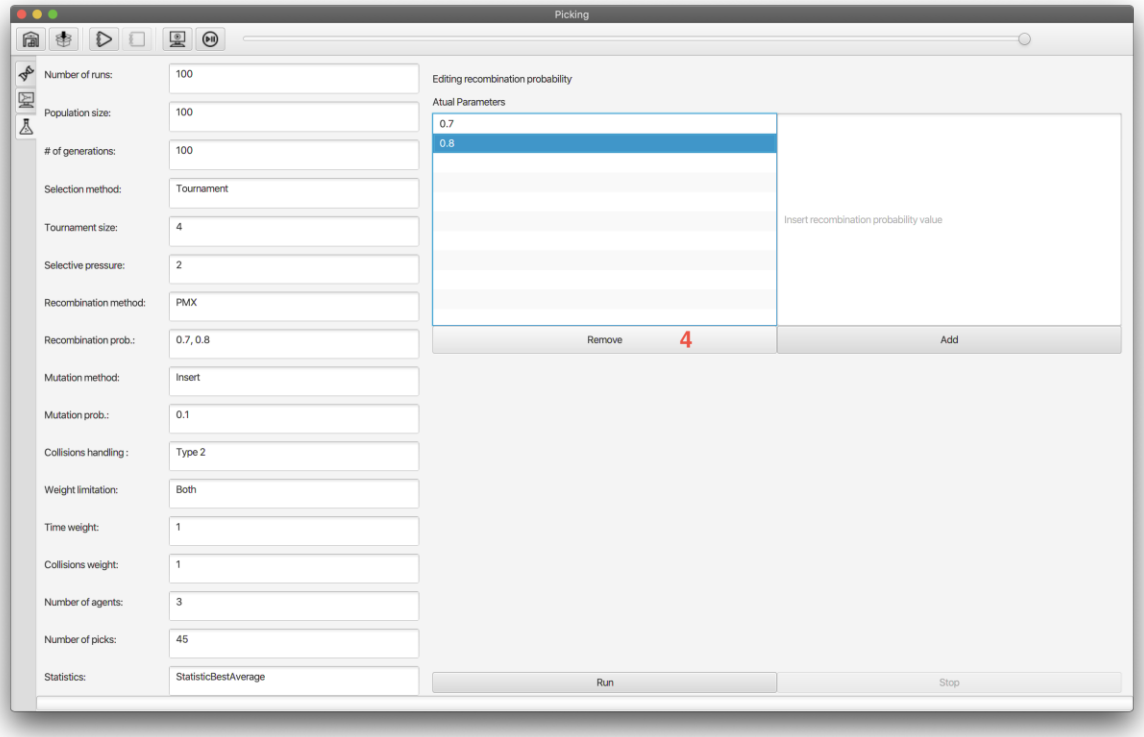

Picking
Seed: 1
Population size: 100
# of generations: 100
Selection method: Tournament
Tournament size: 4
Selective pressure: 2.0
Recombination method: PMX
Recombination prob.: 0.8
Mutation method: Insert
Mutation prob.: 0.1
Collisions handling Type1
Weight limitation Picks
Time weight: 1
Collision weight: 1
Best solution:
Fitness: 152.0
Time: 152.0
Number of Collisions: 0
Collision Wait Time: 0.0
Number of Time Agents Went to Offload: 2
Agent 1 path time : 146.0
Agent 2 path time : 151.0
Agent 3 path time : 152.0
Evolution
fitness
generation
Best
Average

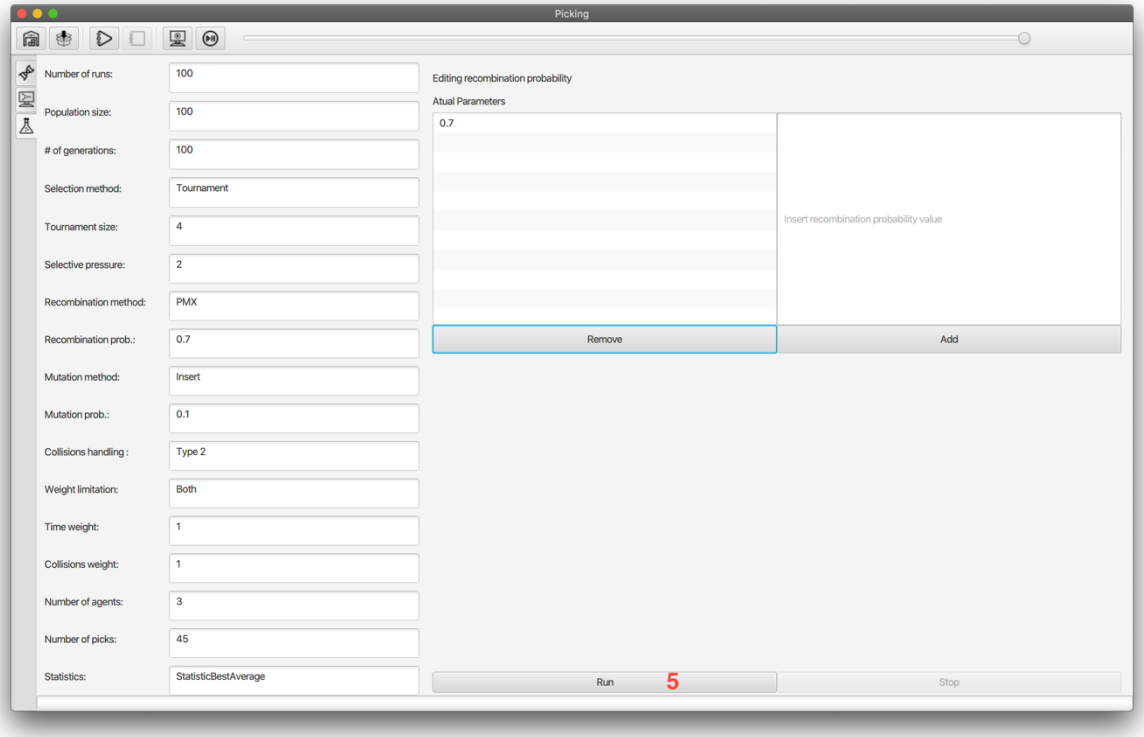

Picking

Executar as experiências

1. Selecionar separador das experiências;
2. Selecionar um parâmetro;
3. Adicionar valores aos parâmetros das experiências;
4. Remover valores aos parâmetros das experiências;
5. Importar o armazém a partir do ficheiro (WarehouseLayout.json) e executar as experiências.




Picking
Number of runs: 100
Population size: 100
# of generations: 100
Selection method: Tournament
Tournament size: 4
Selective pressure: 2
Recombination method: PMX
Recombination prob.: 0.7, 0.8
Mutation method: Insert
Mutation prob.: 0.1
Collisions handling : Type 2
Weight limitation: Both
Time weight: 1
Collisions weight: 1
Number of agents: 3
Number of picks: 45
Statistics: StatisticBestAverage
Editing recombination probability
Atual Parameters
0.7
0.8
Insert recombination probability value
Remove
4
Add
Run
Stop


Picking
Number of runs: 100
Population size: 100
# of generations: 100
Selection method: Tournament
Tournament size: 4
Selective pressure: 2
Recombination method: PMX
Recombination prob.: 0.7
Mutation method: Insert
Mutation prob.: 0.1
Collisions handling : Type 2
Weight limitation: Both
Time weight: 1
Collisions weight: 1
Number of agents: 3
Number of picks: 45
Statistics: StatisticBestAverage
Editing recombination probability
Atual Parameters
0.7
Insert recombination probability value
Remove
Add
Run
5
Stop