Universidade do Minho
Escola de Engenharia

Daniel Oliveira Sousa

Desenvolvimento de Algoritmos para Recolha de Produtos num Armazém Automático de Alta Densidade

Dissertação de Mestrado

Mestrado em Engenharia de Sistemas

Trabalho realizado sob a orientação de

Professor Doutor José António Vasconcelos Oliveira

Setembro de 2020

AGRADECIMENTOS

Na realização deste projeto de dissertação de mestrado contei com o apoio de diversas pessoas cujo contributo foi fundamental para a conclusão do mesmo. A todas elas, o meu sincero agradecimento.

De modo particular, quero agradecer ao Professor Doutor José António Oliveira pela orientação ao longo do projeto, em especial pela atenção e disponibilidade demonstrada para esclarecer as minhas dúvidas durante as nossas longas reuniões. Sem a sua orientação não teria sido possível realizar a presente dissertação.

Agradeço à Pinto Brasil S.A. a oportunidade concedida para estudar um sistema de armazenamento inovador e, em particular, ao Engenheiro Leonel pelo suporte dado ao longo deste projeto.

Agradeço profundamente aos meus pais todo o apoio, confiança e pela oportunidade que me proporcionaram de prosseguir os meus estudos no ensino superior.

Agradeço ao Hilário Sousa a ajuda e o suporte dado ao nível gráfico e técnico ao longo deste projeto de dissertação.

Aos meus amigos e colegas do mestrado agradeço a ajuda e todos momentos incríveis vividos durante o meu percurso académico.

A todos, o meu sincero agradecimento!

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

RESUMO

A presente dissertação realizou-se no âmbito do Mestrado em Engenharia de Sistemas da Universidade do Minho, inserida em contexto industrial na Pinto Brasil S.A., empresa dedicada à metalomecânica. Este projeto surgiu da necessidade de estudar o desempenho de um armazém automático de elevada densidade denominado Shopstocker.

Os sistemas de armazenamento desempenham um papel fundamental na performance de toda a cadeia de abastecimento. A necessidade de adoção de políticas consentâneas com a Indústria 4.0, tais como *Just-inTime*/ *Just-in-Sequence*, levou a que diversos tipos de armazéns fossem desenvolvidos. Um dos mais estudados e implementados é o *Automated Storage and Retrieval System* de elevada densidade. A Pinto Brasil S.A desenvolveu e implementou numa empresa fornecedora de para-choques, um sistema automatizado de elevada densidade para armazenamento de produto acabado. O Shopstocker implementado tem capacidade para armazenar um elevado número de para-choques e entrega os produtos de acordo com uma ordem pré-definida no tempo adequado. Devido à elevada variedade de referências de para-choques que atualmente são produzidos, o armazenamento do Shopstocker gera situações caóticas prejudicando a taxa de extração de produtos. No processo de extração a seleção de referências para uma ordem de carregamento é atualmente um procedimento muito complexo.

Com o estudo da situação real do armazém implementado, modelou-se matematicamente um problema de otimização combinatória e implementou-se o modelo em linguagem de programação Python. Esta metodologia permitiu desenvolver três modelos de otimização, MPA, MF e MTFM, e dois algoritmos adicionais que permitem avaliar o desempenho do Shopstocker. Com os modelos e algoritmos desenvolvidos realizaram-se experiências computacionais de modo a avaliar o impacto que a configuração do armazém e as sequências de extração (encomendas) têm na eficiência do Shopstocker, nomeadamente nas medidas de desempenho, tempo total de extração, percentagem da regra *First-In-First-Out* cumprida e número de produtos movimentados. Os resultados demonstraram que o Shopstocker implementado apresenta uma perda de eficiência devido à elevada variedade de referências disponibilizadas. Assim, é recomendado que a empresa reduza o número de modelos disponibilizados, ou aumente o número de linhas de armazenamento, o que permitiria melhorar de forma significativa os valores das medidas de desempenho estudadas e, consequentemente, a eficiência do Shopstocker. O modelo desenvolvido neste projeto pode ainda ser implementado num Sistema de Apoio à Decisão e fornecerá uma solução otimizada para cada encomenda que chegue ao Shopstocker.

PALAVRAS-CHAVE

Indústria 4.0; Armazém; Otimização; MILP; Python.

ABSTRACT

This dissertation was carried out as part of the master's degree in Systems Engineering at the University of Minho, and in industrial environment at Pinto Brasil S.A., a company dedicated to metalworking. This project emerged from the need to study the performance of a high-density automated warehouse called Shopstocker.

The storage systems play a decisive role in the supply chain performance. The implementation of the Industry 4.0 policies, such as Just-in-Time/Just-in-Sequence, led to the development of several types of warehouses. One of the most studied and implemented is the high-density Automated Storage and Retrieval System. Pinto Brasil S.A. developed and implemented an automated warehouse of finished product in a bumper supplier company. The implemented Shopstocker can store many bumpers and delivers the products by a specific sequence at the right time. Due to the wide variety of bumper references currently provided, the Shopstocker storage process often results in a chaotic situation harming the product extraction rate. In the extraction process, selecting references for a loading order is a very complex procedure.

The actual situation of the implemented warehouse was studied, and a combinatorial optimization problem was modelled and implemented with the Python programming language. This methodology allowed the development of three optimization models, MPA, MF and MTFM, and two additional algorithms that enable to evaluate the Shopstocker's performance. Computational experiments were performed with the developed models to evaluate the impact that warehouse configuration and extraction sequences (orders) have in the Shopstocker's efficiency, namely in the key performance indicators, total extraction time, percentage of the First-In-First-Out rule fulfilled and the number of moved bumpers. The results showed that the implemented Shopstocker has a loss of efficiency due to the wide variety of references currently provided. According to this study, it is recommended that the company reduce the number of references provided, or increase the number of storage lines, which will allow to improve the key performance indicators values and, consequently, the Shopstocker's efficiency. The developed model can also be implemented in a Decision Support System and will provide an optimized solution to each order that arrive at Shopstocker.

KEYWORDS

Industry 4.0; Warehouse; Optimization; MILP; Python.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÓNIMOS

AGV – *Automated Guided Vehicles*

AML – *Algebraic Modeling Languages*

AS/RS – *Automated Storage and Retrieval System*

CA – Cadeia de Abastecimento

COI – *Cube-per-Order Index*

DOS – *Duration-of-Stay*

FIFO – *First-In-First-Out*

I/O – *Input/Output*

IoE – *Internet of Everything*

IoP – *Internet of People*

IoS – *Internet of Services*

IoT – *Internet of Things*

JIT – *Just-In-Time*

JIS – *Just-In-Sequence*

KPI – *Key Performance Indicator*

MF – Modelo FIFO

MPA – Modelo Problema de Afetação

MTFM – Modelo Tempo FIFO Movimentados

OCA – Orientação da Cadeia de Abastecimento

PB – Pinto Brasil

PC – Para-choques

PPR-SL – *Percentage Priority to Retrievals with the Shortest Leg*

Pyomo – *Python Optimization Modeling Objects*

SCM – *Supply Chain Management*

SDC – *Shortest Dual-command Cycle*

S/R – *Storage and Retrieval*

TSP – *Travelling Salesman Problem*

WIP – *Work-in-Process*

1. INTRODUÇÃO

A presente dissertação foi realizada no âmbito do Mestrado em Engenharia de Sistemas. O projeto desenvolveu-se em ambiente empresarial, na Pinto Brasil S.A..

Neste capítulo é feito um enquadramento do projeto de dissertação e dos temas tratados. Posteriormente, são descritos os diferentes objetivos aos quais esta dissertação se propôs a dar resposta. Por fim, é descrita a metodologia utilizada e é apresentada a estrutura da dissertação.

1.1 Enquadramento

Nas últimas décadas, a maioria das empresas começou a considerar o conceito e implementação da Gestão da Cadeia de Abastecimento, em inglês *Supply Chain Management* (SCM), como a disciplina estratégica de maior importância para a sua sobrevivência organizacional e vantagem competitiva. Para Ross (2010) o reconhecimento da importância do SCM por parte das organizações deveu-se, sobretudo, à perceção de que a sua capacidade para reinventar continuamente a vantagem competitiva, depende menos das suas competências internas e mais da capacidade para olharem para os seus parceiros de negócio. Desta forma, os diferentes colaboradores da cadeia de abastecimento podem unir as suas capacidades com o intuído de melhorarem os seus processos, produtos e estratégias (Ross, 2010).

De acordo com Pfohl, Yahsi, & Kuznaz (2015) as inovações disruptivas estão a alterar a visão estratégica de muitas empresas e dos seus modelos de negócio. O aumento dos processos digitalizados e o crescimento exponencial dos sensores, que permitem a recolha de dados, tem levado a que a cadeia de abastecimento esteja a experienciar o impacto da Indústria 4.0, também conhecida como a quarta revolução industrial (Pfohl et al., 2015). A Indústria 4.0 é caracterizada pela mudança de paradigma de controlo centralizado para controlo descentralizado dos processos produtivos, associada com a implementação de novas tecnologias, especialmente ligadas à internet. No futuro, os produtos inteligentes irão conhecer todo o seu histórico produtivo, o posto em que se encontram e o seu destino, e irão, de forma autónoma, instruir as máquinas a realizarem os processos de fabricação requeridos e, aos transportadores, a movimentação necessária até à estação seguinte (Hermann, Pentek, & Otto, 2016).

Grandes, médias e até mesmo pequenas empresas do setor automóvel, estão a sofrer um rápido desenvolvimento em termos tecnológicos e de aplicação de sistemas inteligentes, em resultado da grande mudança na procura do mercado global. Com o crescimento das organizações, a necessidade de

sistemas de transporte e armazenamento mais eficientes aumenta, em especial na área da fabricação (Halim, Jaffar, Yusoff, & Adnan, 2012). A crescente pressão do mercado nas empresas obriga a que estas, para se manterem competitivas no seu setor, tenham de apostar na inovação dos seus produtos de forma a melhorar o controlo de custos e a flexibilidade na resposta personalizada que o cliente procura. As organizações do setor industrial veem-se obrigadas a tolerar a produção de pequenos lotes, ou à peça, para satisfazer de forma rápida e eficaz a procura existente (Verebová, 2016). Para garantir esta versatilidade, o controlo e implementação de operações logísticas mais eficientes em toda a cadeia de abastecimento é, por isso, determinante. Para Fusko, Rakyta, & Manlig (2017) a logística é um fator decisivo na competitividade e economia das organizações. A evolução para um mercado dinâmico, causado principalmente pela globalização, exige requisitos e soluções logísticas de elevada eficiência. Para atingir estes objetivos, as instalações produtivas e, principalmente, os sistemas intralogísticos, desempenham um papel fundamental no fluxo adequado de materiais e informação dentro das organizações. A premência de estratégias de entrega dos produtos de acordo com as filosofias *Just-In-Time* e *Just-in-Sequence*, (JIT/JIS) requer um funcionamento eficiente de toda a cadeia de abastecimento e, em particular, dos sistemas de armazenamento.

Apesar da tendência de redução de inventários, assim como de todas as tarefas que não acrescentam valor ao produto, em algumas situações a utilização de armazéns é inevitável (Dolgui & Proth, 2010). A definição da forma mais eficiente e de menor custo para realizar as operações de armazém, tais como, receção, armazenamento, *picking*, entre outras, é, por isso, fulcral. A constante pesquisa e evolução no contexto dos armazéns levou a que diversos sistemas e tecnologias fossem desenvolvidos. Os armazéns automatizados e de alta densidade, em especial os AS/RS (*Automated Storage and Retrieval System*), são cada vez mais utilizados pelos centros produtivos e de distribuição. O objetivo destes sistemas de armazenamento é garantir o controlo de inventário, a otimização dos recursos e o cumprimento das filosofias JIT/JIS (Manzini, 2012). Segundo Gong, Zhang, & Wang (2009) as operações de *picking*, representam cerca de 55% dos custos totais dos processos de armazém. Por esta razão, mesmo em sistemas automatizados, a otimização do processo de recolha é fundamental para o cumprimento dos tempos de entrega ao cliente da forma mais eficiente e de menor custo possível.

De modo a contribuir para uma logística mais eficiente, a Pinto Brasil S.A. (PB) dedica-se ao desenvolvimento de soluções logísticas inovadoras para o setor automóvel. Uma solução logística criada pela PB consiste num armazém automático de elevada densidade de produto acabado que designou por Shopstocker. Recentemente, a PB implementou um Shopstocker na empresa fabricante de para-choques (PC) no setor automóvel que neste projeto será identificada como CLxPB. O Shopstocker é um AS/RS

que combina o armazenamento automatizado com tecnologias de transporte aéreo. Este sistema de armazenamento proporciona um controlo integral do processo de armazenamento e expedição dos para-choques, desde a sua chegada ao sistema, até à entrega ao cliente nas condições pretendidas.

A Figura 1 apresenta um esquema simplificado representativo do Shopstocker. Como é possível observar na figura, existe uma zona de *input*, onde os PC são introduzidos no sistema. Após a inserção os PC são tracionados até às linhas de armazenamento onde permanecem até que seja solicitada a sua movimentação. No interior da linha de armazenamento os PC fluem através da força gravitacional desde a extremidade de inserção até à extremidade de extração. Quando solicitados para uma sequência, os PC são extraídos da linha de armazenamento e tracionados até à zona de *output*.

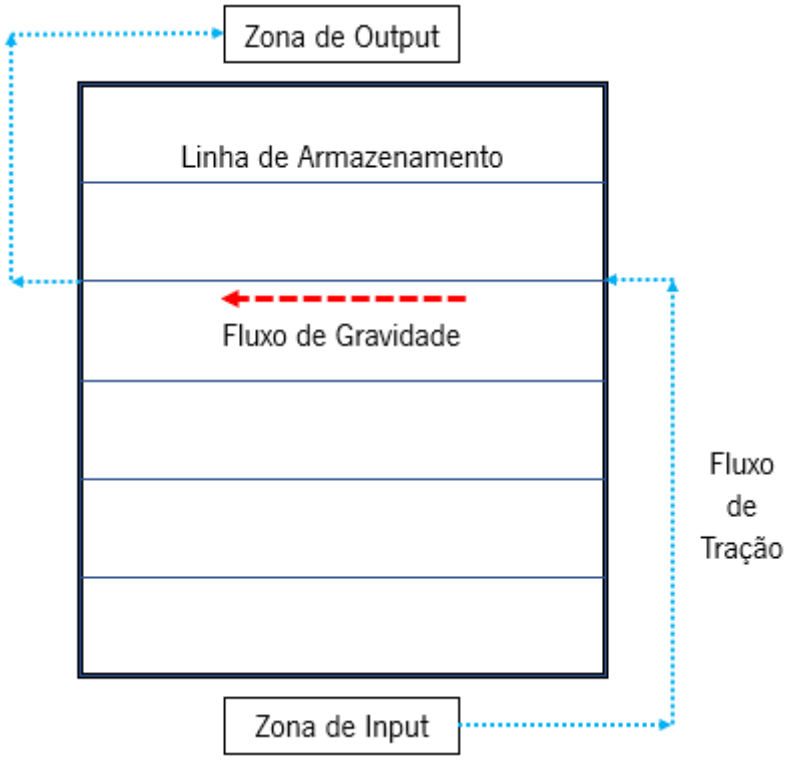

Figura 1 - Esquema simplificado representativo do Shopstocker.

O armazenamento no Shopstocker segue uma política baseada em três classes: *high runners*, *medium runners* e *low runners*. Nas linhas dedicadas aos *high runners* só é permitida a inserção de produtos de uma referência. Já nas linhas dedicadas a armazenar *medium runners* é possível conciliar até três modelos distintos. Por fim, as linhas dedicadas aos *low runners* são linhas caóticas, não tendo limite de referências. Esta política tem implicações no processo de extração dos para-choques. Sempre que uma encomenda chega ao sistema, com as referências solicitadas e a sequência de carregamento

pretendida, o Shopstocker seleciona os para-choques que estão há mais tempo no sistema cumprindo a regra FIFO (*First In Fisrt Out*). Nas linhas dedicadas a *high runners* a seleção dos PC a abandonar o sistema é imediata. Nas restantes linhas com múltiplas referências, o processo de seleção é mais complexo. O sistema precisa de analisar, para cada opção, a posição do para-choques na linha, o seu tempo de permanência no sistema e a distância até à zona de saída, garantindo o *takt time*.

Com o intuito de modelar o funcionamento do Shopstocker no processo de seleção de produtos para expedição, foi necessário identificar o problema de otimização combinatória associado à escolha do melhor conjunto de PC a ser extraído do armazém e encontrar algoritmos adequados que fossem capazes de o resolver. O presente projeto de dissertação visou dar resposta à seguinte pergunta de investigação: O atual modo de seleção de produtos para o *picking* no Shopstocker é adequado ao cumprimento do *takt time* e da regra FIFO?

1.2 Objetivos

Este projeto de dissertação teve como principal objetivo o desenvolvimento e a avaliação de regras de seleção de produtos adequadas ao cumprimento do *takt time* e seleção FIFO, de forma a que o processo de *picking* no armazém automatizado estudado fosse realizado da forma mais eficiente e de menor custo possível.

Para além do objetivo geral, este projeto propôs-se a cumprir os seguintes objetivos secundários:

i) Análise da situação atual do armazém e identificação dos KPI's (*Key Performance Indicator*) importantes para a avaliação do armazém;

ii) Criação de um simulador de políticas de seleção de produtos para *picking*;

iii) Definição de layouts representativos do estado atual do armazém;

iv) Desenvolvimento de algoritmos de seleção de produtos para o *picking*;

v) Estudo do impacto nos KPI's das diferentes alternativas de seleção de produtos para o *picking*.

1.3 Metodologia

O presente projeto de dissertação seguiu uma metodologia amplamente utilizada e descrita na literatura, a investigação-ação. Esta metodologia foi dividida em várias etapas por Lewin, K (1946). Segundo o autor, após a identificação de um problema é necessário, numa primeira fase, planear qual o melhor processo para o resolver. Após a conclusão desta etapa surgem dois tópicos centrais: um plano

global para se atingir o objetivo final e uma decisão relativa ao primeiro passo da ação. Posteriormente, executa-se a primeira atividade definida, e avaliam-se as diferentes consequências que advêm da sua implementação no sistema real. Assim é possível avaliar a ação, verificar se os resultados excedem ou ficam aquém das expectativas, obter conhecimento pela observação da situação e, por fim, melhorar e corrigir as ações futuras. Os passos seguintes desta metodologia são realizados de forma cíclica, percorrendo as etapas definidas anteriormente: planeamento, ação e avaliação. O projeto de investigação desenrola-se até que todas as fases delineadas inicialmente sejam percorridas e o objetivo final tenha sido alcançado.

Por sua vez, Clark & Trist (1976) afirmaram que a metodologia de investigação-ação proporciona a descoberta de novos factos científicos e, em simultâneo, resolve problemas práticos. Para os autores, na base desta metodologia está o paradigma científico tradicional da manipulação experimental e da observação dos seus efeitos. Contudo, ao invés da realização da experiência de forma isolada pelo investigador, há colaboração entre os diferentes interessados no processo. O modelo desenvolvido por Clark & Trist (1976) atribui menor importância à fase de planeamento, e privilegiava a fase de avaliação, tanto do problema inicial, como das ações implementadas. Desta forma, os autores procuraram aumentar a liberdade e o crescimento da investigação, favorecendo a pesquisa como ação em detrimento da resolução de problemas individuais e da mudança de comportamentos.

Mais recentemente, Stringer (2007) definiu a metodologia de investigação-ação como uma abordagem sistemática de pesquisa que permite encontrar soluções efetivas para problemas existentes no nosso quotidiano. O autor defende ainda que, ao contrário das metodologias de investigação experimentais e quantitativas, que apenas analisam um número reduzido de variáveis e oferecem explicações generalizadas, a investigação-ação procura envolver as dinâmicas complexas existentes em qualquer contexto social. Para além disso, esta metodologia utiliza ciclos contínuos de pesquisa, concebidos para encontrar soluções para problemas específicos e localizados, providenciando assim, formas para as organizações educativas ou empresariais aumentarem a sua eficiência. O objetivo central é, por isso, obter novo conhecimento que permita às pessoas envolvidas melhorar o seu bem-estar, através do aumento da qualidade dos seus processos e tarefas (Stringer, 2007).

Desta forma, com a metodologia de investigação-ação em foco, este projeto de dissertação passou essencialmente por seis etapas. Numa primeira fase procedeu-se ao estudo e levantamento da situação atual do Shopstocker, ao nível de colocação e recolha de para-choques, e à identificação dos KPI's mais relevantes. Posteriormente, foi necessário desenvolver um modelo representativo da operação de recolha de para-choques. A terceira tarefa deste projeto de dissertação consistiu na definição e caracterização

de um problema de otimização combinatória. Após a realização destas etapas, o projeto entrou na fase de implementação em linguagem de programação do problema de otimização combinatória definido. Com o intuito de avaliar e validar os algoritmos desenvolvidos na fase anterior, foi necessário realizar experiências computacionais e estudar os KPI's selecionados. Por fim, com o todos os elementos necessários previamente tratados, procedeu-se à escrita da presente dissertação.

1.4 Estrutura da Dissertação

A presente dissertação encontra-se dividida em 6 capítulos. O primeiro e atual capítulo serve de introdução ao projeto desenvolvido, no qual é feito um enquadramento ao problema real estudado e são descritos os objetivos e a metodologia de investigação utilizada.

No Capítulo 2 é apresentada a revisão bibliográfica dos principais temas tratados ao longo deste projeto tais como: gestão da cadeia de abastecimento, Indústria 4.0 e armazéns. Neste capítulo é dada especial enfâse ao estado de arte dos armazéns automatizados, em especial os de alta densidade.

No terceiro capítulo da presente dissertação é apresentada a empresa e é descrito o sistema real de armazenamento estudado, o Shopstocker. Neste capítulo identificam-se ainda os problemas existentes no armazém e as medidas de desempenho essenciais para a análise do funcionamento do mesmo.

No quarto capítulo descreveu-se de forma exaustiva a metodologia utilizada neste projeto. Numa fase inicial descreveu-se o processo de análise e geração de dados. Na segunda parte deste capítulo encontra-se o modelo matemático desenvolvido e a sua implementação em linguagem de programação Python.

No quinto capítulo são apresentadas as experiências computacionais realizadas com os modelos e as instâncias desenvolvidas e são apresentadas as causas identificadas para os efeitos observados através dos resultados obtidos.

No Capítulo 6 são apresentadas as principais conclusões deste projeto de dissertação. São ainda feitas sugestões de investigação futura para estudo neste tipo de sistemas de armazenamento.

2. REVISÃO DA LITERATURA

Neste capítulo é apresentada uma revisão dos principais temas e conceitos utilizados neste projeto de dissertação tais como: gestão da cadeia de abastecimento, Indústria 4.0 e armazéns. Na última parte deste capítulo é dada especial enfâse ao estado de arte dos armazéns automatizados, em especial os de elevada densidade.

2.1 Gestão da Cadeia de Abastecimento

2.1.1 Definição de Cadeia de Abastecimento

Handfield & Nichols Jr. (1999) definiram a cadeia de abastecimento (CA) como sendo o conjunto de todas as atividades associadas à movimentação e transformação de bens, desde a fase de extração das matérias primas até ao consumidor final, incluindo o fluxo de informação. Já para Mentzer et al. (2001), a CA pode ser vista em três níveis distintos, direta, extensa e final. A CA direta envolve apenas o fornecedor e o cliente direto da empresa. Já na CA extensa são incluídos os fornecedores do fornecedor direto e os clientes dos clientes diretos da empresa. Por sua vez, a CA final engloba todos os fornecedores e clientes da empresa, desde as matérias primas até ao consumidor final. A CA pode por isso ser definida, de uma forma mais abrangente, como o conjunto de todos os processos e intervenientes responsáveis pela obtenção de um produto ou serviço, desde a extração de matérias primas até ao cliente final.

2.1.2 História da Gestão da Cadeia de Abastecimento

A gestão da cadeia de abastecimento, surgiu segundo Ross (2010), com a evolução da logística que se iniciou no século passado. A passagem das decisões logísticas dos departamentos de vendas, financeiro e de produção para um departamento próprio possibilitou uma melhor gestão dos custos associados a todas as atividades logísticas envolvidas. A centralização da logística, associada com a incorporação do computador, foi determinante para que, por volta de 1980, surgissem algumas preocupações, tais como o JIT, o controlo da qualidade e o serviço ao cliente. Nos anos que se seguiram, a elevada concorrência e pressão do mercado, assim como as novas filosofias de gestão desenvolvidas, levaram a que competição e controlo da qualidade se tornassem as palavras de ordem para as empresas. A compreensão da importância das vias de abastecimento para a competitividade das organizações originou a introdução de um novo termo, Logística Integrada. As empresas perceberam que os esforços

conjuntos dos diferentes parceiros resultavam numa redução de custos e, consequentemente, numa vantagem competitiva. Em meados dos anos noventa, a aceleração da globalização, o aumento do poder de decisão do cliente sobre os produtos, a terceirização e o desenvolvimento das tecnologias de informação, levaram ao surgimento do SCM. Mais recentemente, a aplicação das tecnologias de informação, como a Internet, revolucionou a forma como as empresas gerem a sua cadeia de abastecimento, evoluindo para um sistema colaborativo, e em tempo real, entre os diferentes parceiros da cadeia (Ross, 2010).

A Figura 2, ilustra as diferenças e as vantagens para a CA com a introdução do SCM, apresentadas por Mentzer et al. (2001).

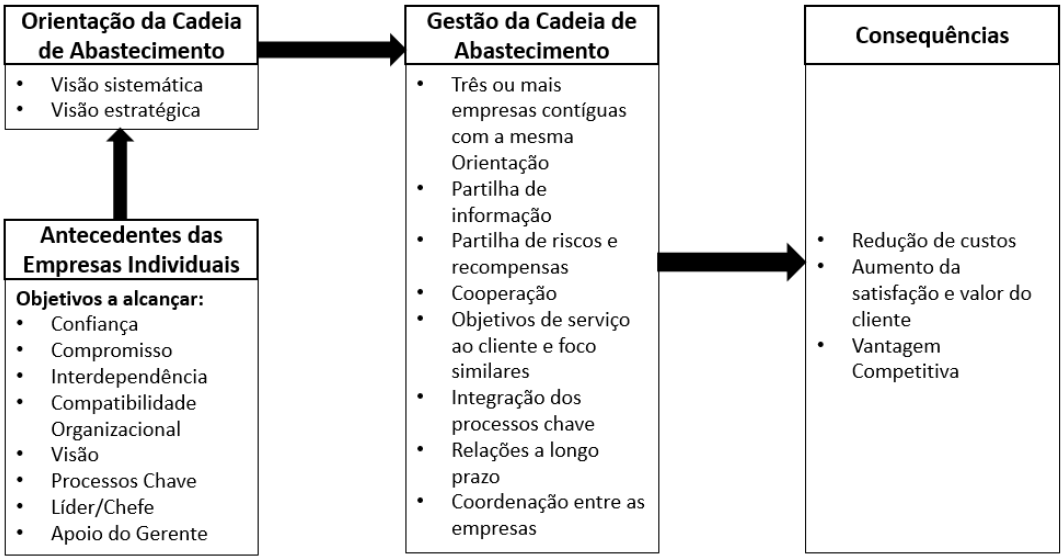

Figura 2 - Implicações do SCM. Figura adaptada de Mentzer et al. (2001).

Os valores e interesses individuais das empresas, tais como, confiança, visão, processos chave e a orientação do gerente, foram combinados de modo a existir uma visão sistemática e estratégica comum de toda a CA, denominada Orientação da Cadeia de Abastecimento (OCA). Com a implementação do SCM, as organizações da CA passaram a cooperar entre si, partilhando informação, riscos e recompensas, integraram os seus processos chave e uniram os seus objetivos de modo a cumprir a orientação conjunta definida. A Figura 2 realça ainda as consequências esperadas após a implementação do SCM, tais como, a redução de custos, o aumento da satisfação do cliente e o aumento da vantagem competitiva de toda a CA (Mentzer et al., 2001).

2.1.3 Definição de Gestão da Cadeia de Abastecimento

Mentzer et al. (2001) investigou e apresentou seis definições diferentes para o SCM, recolhidas de diversos autores, e constatou que se enquadravam em três categorias: filosofia de gestão, implementação de uma filosofia de gestão e o conjunto de processos de gestão. Através do estudo realizado, o autor definiu o SCM como a coordenação sistémica e estratégica das funções de negócio tradicionais, e das suas táticas, dentro de uma determinada empresa, e desta com a sua CA, com o objetivo de melhorar o desempenho a longo prazo de cada uma das empresas a nível individual e, mais importante, da CA como um todo (Mentzer et al., 2001). O modelo proposto por esta definição pode ser visualizado através da Figura 3, que evidencia os fluxos existentes na CA, as atividades em que é necessário cooperação entre as diferentes organizações e, de forma destacada na extremidade da seta, os objetivos finais que se pretendem alcançar através da implementação do SCM, nomeadamente, satisfação do cliente, valor, lucro e vantagem competitiva.

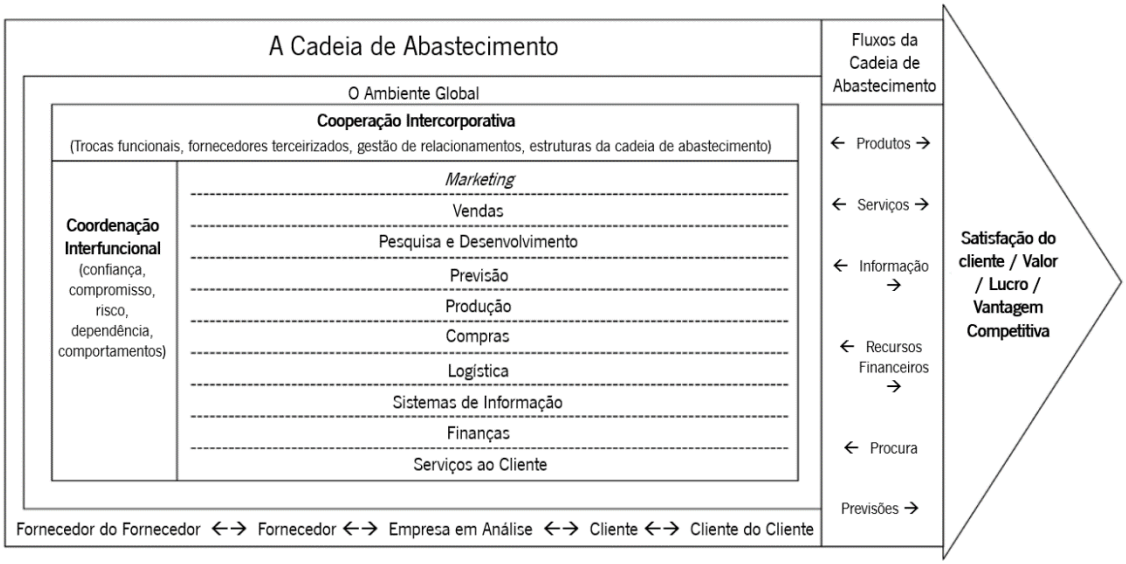

Figura 3 - Modelo conceptual do SCM. Imagem adaptada de Mentzer et al. (2001).

Para Ross (2010) o SCM pode ser definido, de uma forma mais sucinta, como a capacidade das empresas integrarem as suas capacidades e processos com os seus fornecedores e clientes de forma estratégica, com o intuito de alcançarem objetivos comuns.

2.2 Indústria 4.0

2.2.1 Origem do termo Indústria 4.0

Ao longo dos últimos trezentos anos, a indústria sofreu grandes avanços que transformaram completamente os seus processos. Em 1760, o surgimento e introdução da máquina a vapor alterou drasticamente os processos produtivos. Este avanço iniciado no Reino Unido ficou conhecido como a primeira revolução industrial (Hermann et al., 2016). Mais tarde, a partir de 1870, a eletrificação, a incorporação dos motores de combustão e as linhas de montagem inovadoras conduziram à segunda revolução industrial (Pfohl et al., 2015). A terceira revolução industrial iniciou-se por volta de 1970 com a grande evolução dos dispositivos eletrónicos e das tecnologias de informação, que permitiram o aumento da automatização dos processos industriais (Mentzer et al., 2001).

Nas últimas décadas, a grande evolução tecnológica e a procura de produtos e serviços cada vez mais sofisticados por parte dos consumidores, tem colocado novos desafios e oportunidades para a indústria (Barreto, Amaral, & Pereira, 2017). Em 2011, a Alemanha, com o intuito de se manter na vanguarda do setor industrial, introduziu publicamente o termo Indústria 4.0, numa feira comercial em Hanôver (Hofmann & Rüsch, 2017; Lasi, Fettke, Kemper, Feld, & Hoffmann, 2014). A Indústria 4.0, também conhecida como quarta revolução industrial, procura tirar proveito da explosão do desenvolvimento das tecnologias de informação associadas à internet, tais como, sistemas ciber físicos, *Internet of Things* (IoT), *Internet of Services* (IoS) e *Internet of Everything* (IoE), para que as empresas se adaptem às novas condições de mercado, tornando-se *Smart Factories* (Hofmann & Rüsch, 2017).

2.2.2 Definição de Indústria 4.0

Desde a introdução do termo em 2011, a Indústria 4.0 não conseguiu uma definição consensual. Ao longo da última década, diversos autores investigaram e procuraram uniformizar o conceito desta nova filosofia (Barreto et al., 2017; Hermann et al., 2016; Hofmann & Rüsch, 2017; Luthra & Mangla, 2018; Pfohl et al., 2015).

Pfohl et al. (2015) definiu a Indústria 4.0 como a soma de todas as inovações disruptivas implementadas na CA que seguem as tendências: digitalização, automatização, transparência, mobilidade, modulação, colaboração em rede e socialização dos produtos e serviços.

Mais tarde, Hermann et al. (2016) apresentou uma revisão da literatura, referente à Indústria 4.0, procurando encontrar os termos e expressões mais associados a esta nova filosofia. O autor identificou

quatro princípios fundamentais que são comuns à grande maioria dos artigos estudados, interconexão, transparência da informação, decisões descentralizadas e assistência técnica.

Hofmann & Rüsch (2017) publicaram um artigo focado na influência da Indústria 4.0 na logística das empresas. A revisão feita pelos autores permitiu que definissem esta nova filosofia como uma mudança na lógica da fabricação no sentido do aumento da descentralização e da abordagem autorreguladora da criação de valor, sustentada pela introdução de sistemas ciber físicos, da IoT, da IoS, do *cloud computing* e das *smart factories*, que irá permitir às empresas responder aos requisitos atuais da produção.

Já para Luthra & Mangla (2018), a Indústria 4.0 pode ser entendida como a aplicação de novas tecnologias de informação e comunicação integradas com a automação industrial, com as redes de dados e com as novas tecnologias de fabricação, tais como, produção inteligente, interação humano-computador, impressão 3D, entre outras.

Apesar dos numerosos estudos já realizados, uma definição mais clara e consensual da Indústria 4.0 e dos seus princípios é necessária, para que, a discussão científica do tema seja possível e a sua adoção pelas empresas seja uma realidade (Hermann et al., 2016).

2.2.3 Princípios da Indústria 4.0

Das diversas publicações que têm procurado definir os princípios e componentes essenciais da Indústria 4.0, há duas que se destacam na literatura: a de Lasi et al. (2014) que descreve as duas forças motoras da quarta revolução industrial, e a de Hermann et al. (2016) que estabelece os quatro princípios que definem a Indústria 4.0.

Lasi et al. (2014) publicou um artigo dedicado à Indústria 4.0 onde indicou as duas forças motoras da quarta revolução industrial: *application-pull* e *technology-push*. Para o autor, o *application-pull* irá induzir mudanças significativas a nível económico, social e político, tais como:

- Períodos de desenvolvimento mais curtos: a capacidade de inovação rápida das empresas é um fator determinante para a sua competitividade.
- Individualização da procura: as condições e requisitos de um negócio começam cada vez mais a ser determinados pelos compradores, ao invés dos vendedores, o que obriga a uma maior personalização.
- Flexibilidade: uma maior versatilidade, especialmente nas áreas produtivas, é um requisito de grande importância para as empresas

- Descentralização: é necessário reduzir as hierarquias organizacionais para que o processo de tomada de decisão seja mais rápido e ajustado às condições em tempo real.
- Eficiência dos recursos: a escassez, os elevados preços e as preocupações ecológicas, obrigam a uma maior sustentabilidade no contexto industrial.

Para além das mudanças que as aplicações estão a induzir, Lasi et al. (2014) indicou ainda o forte impulso tecnológico que já está a alterar o funcionamento diário do contexto empresarial:

- Aumento da automatização e mecanização: surgem cada vez mais ferramentas de auxílio ao trabalho físico e os processos repetitivos serão totalmente automatizados.
- Digitalização e trabalho em rede: o aumento da utilização de sensores vai permitir a recolha de maior volume de dados que, posteriormente, possibilitarão maior análise e controlo da atividade industrial.
- Miniaturização: a tendência para reduzir o volume de todo o tipo de aparelhos, liberta mais espaço para novas áreas de aplicação.

Numa outra perspetiva, Hermann et al. (2016) definiu os quatro princípios que, no seu entender, definem a Indústria 4.0: interconexão, transparência da informação, decisões descentralizadas e assistência técnica. A interconexão das pessoas, máquinas, aparelhos e sensores, é possível através da IoT e IoP que combinados formam a IoE, e permitem a partilha de informação entre os diversos intervenientes, com o intuito de se alcançarem os objetivos comuns. A transparência da informação será alcançada com a fusão entre os sistemas físicos e virtuais, e irá permitir um processo de tomada de decisão mais rápido e robusto. Da mesma forma, para se alcançar a descentralização, outro dos pilares da Indústria 4.0, as empresas têm de estar aptas a partilhar a informação para além das suas próprias instalações. A globalização da informação por toda a CA, permitirá encurtar o tempo de resposta às perturbações existentes em qualquer ponto da cadeia, possibilitando um aumento da produtividade geral. A mudança das funções humanas de operadores de máquinas para decisores ou funções de manutenção obriga a que a informação chegue de forma compreensível e em tempo-real através de sistemas de assistência técnica. A utilização de robots em funções indesejadas pelo ser humano é uma outra forma de assistência técnica que a Indústria 4.0 procura dar resposta (Hermann et al., 2016).

2.2.4 Implicações da Indústria 4.0

A inovação é hoje sinónimo de progresso e modernidade, e procura encontrar novas soluções que contribuam para a vantagem competitiva no mercado e, consequentemente, para um aumento do nível

de desenvolvimento económico e social e da qualidade de vida (Witkowski, 2017). Do ponto de vista organizacional, os gestores estão a pressionar para a incorporação de novas tecnologias e processos inovadores nas CA que conduzam a uma maior sustentabilidade (Luthra & Mangla, 2018). Os benefícios mais relevantes da implementação da Indústria 4.0 prendem-se com o aumento da flexibilidade, dos padrões de qualidade, da eficiência dos recursos e da produtividade (Tjahjono, Esplugues, Ares, & Pelaez, 2017). Vários autores investigaram as implicações desta nova filosofia para as empresas e para a CA (Barreto et al., 2017; Hofmann & Rüsch, 2017; Lasi et al., 2014; Luthra & Mangla, 2018; Pfohl et al., 2015; Tjahjono et al., 2017).

Lasi et al. (2014) identificou os conceitos fundamentais que irão alterar o funcionamento das organizações:

- *Smart factory*: a fabricação será completamente equipada com sensores e sistemas automatizados, tirando proveito de novas tecnologias inteligentes desenvolvidas.
- Sistemas ciber físicos: a integração do mundo virtual com o mundo físico permitirá controlar todos os processos e produtos físicos através da recolha digital de parâmetros dos diferentes processos.
- Auto-organização: a descentralização, conseguida com o fim das hierarquias de produção, permitirá decisões mais rápidas e eficazes em toda a CA.
- Novos sistemas de procura e distribuição: a procura e a distribuição serão mais individualizadas e novos canais ao longo da cadeia serão gerados.
- Novos sistemas de desenvolvimento de produtos e serviços: a personalização será um requisito cada vez mais necessário e exigido pelos consumidores.
- Adaptação às necessidades humanas: os novos sistemas desenvolvidos deverão responder às necessidades humanas e não o oposto.
- Responsabilidade social corporativa: a sustentabilidade é hoje um fator determinante pelo que, a eficiência dos recursos, reduzindo ao máximo o desperdício, é um dos objetivos da Indústria 4.0.

Tjahjono et al. (2017) realizou um estudo intensivo das consequências da implementação da Indústria 4.0 na CA. O autor concluiu que as áreas que maior impacto sofrerão serão os processos de resposta aos pedidos do cliente e a logística de transportes. Numa outra análise, o autor percebeu que as tecnologias que mais oportunidades geram para as organizações são a realidade virtual e aumentada, impressão 3D e simulação. Por outro lado, algumas tecnologias, tais como, análise do *big data*,

miniaturização, RFID, drones, ciber segurança, entre outros, podem ser benéficos ou ameaças para as organizações, dependendo da forma como são implementadas e utilizadas.

Da mesma forma, Hofmann & Rüsch (2017) concluíram que os produtos e serviços serão cada mais flexíveis e estarão conectados em rede. A automatização e a autorregulação em processos de transporte, produção e processamento das encomendas, serão possíveis com a implementação de sistemas ciber físicos. Os dados vão ser recolhidos ao nível físico e posteriormente analisados ao nível digital. Esta nova perspetiva permitirá aumentar a competitividade de toda a CA.

Apesar das inúmeras vantagens que a implementação da Indústria 4.0 aparenta trazer para toda a CA, os autores concordam que ainda é necessário uma maior investigação e análise dos impactos e desafios que esta nova filosofia terá, quando aplicada em contexto real (Luthra & Mangla, 2018; Tjahjono et al., 2017).

2.3 Armazéns

Os armazéns são estruturas importantes para consolidar produtos de forma a reduzir os custos de transporte, permitir economias de escala, providenciar processos de valor acrescentado, reduzir os tempos de resposta ou para fazer negócio através do aluguer de espaço (Gong et al., 2009). Muitos, senão todos os materiais, têm de passar, num qualquer ponto da CA, por um armazém. A principal função de um armazém prende-se com o armazenamento temporário de bens. Apesar das novas tendências de redução de todo o tipo de operações que não acrescentam valor ao produto, como a eliminação de inventários, em algumas situações a utilização de um armazém é inevitável (Dolgui & Proth, 2010). Ao longo das últimas décadas, diversas investigações e publicações foram feitas na procura de soluções que reduzissem os custos associados aos processos e operações de armazém (Berry, 1968; de Koster, Le-Duc, & Roodbergen, 2007; Dolgui & Proth, 2010; Gong et al., 2009; Gu, Goetschalckx, & McGinnis, 2010; Rouwenhorst et al., 2000). Este interesse, comum a todas as CA, permitiu que fossem desenvolvidos vários tipos de armazém e diversas políticas operacionais, capazes de se adaptarem às necessidades específicas de cada empresa.

2.3.1 Tipos de Armazém

Dolgui & Proth (2010) dividem os armazéns em 4 categorias distintas, baseadas na taxonomia mais comum:

- Armazéns de abastecimento do retalho, que recebem produtos acabados de vários fornecedores dispersos e abastecem as lojas de venda a retalho.
- Armazéns de peças separadas, que recebem componentes de vários fornecedores e, por sua vez, garantem o abastecimento da produção de outros fabricantes, numa fase mais avançada da CA.
- Armazéns de venda por correspondência, que armazenam uma grande variedade de artigos e satisfazem encomendas pequenas, por vezes só de um item, provenientes maioritariamente por via online.
- Armazéns especiais, que são menos vulgares e maioritariamente são alugados para armazenar produtos caros, volumosos e com baixa procura.

Por outro lado, segundo Dolgui & Proth (2010), os armazéns podem também ser classificados de acordo com as suas funções:

- Armazéns privados, que são usados para serviços de armazenamento e abastecimento do interesse dos seus clientes. Os proprietários desempenham funções de distribuição ao longo da CA.
- Armazéns públicos, que podem ser alugados por curtos períodos para lidar com necessidades momentâneas.
- Centros de distribuição, que recebem uma grande quantidade e diversidade de produtos e fornecem aos clientes em pequenos lotes. Este tipo de armazém é muito utilizado na indústria alimentar.
- Armazéns que desempenham serviços de valor acrescentado e fazem parte do sistema de produção. Estes armazéns realizam tarefas como embalamento, etiquetagem e montagem dos produtos armazenados.
- Armazéns que realizam entregas periódicas, que são muito usados em sistemas JIT.
- Armazéns com controlo climático, que são maioritariamente utilizados para armazenar produtos alimentares, garantindo as condições ideais de preservação do produto.

2.3.2 Operações de Armazém

O fluxo dos artigos pelo armazém pode ser dividido em várias etapas, como ilustra a Figura 4.

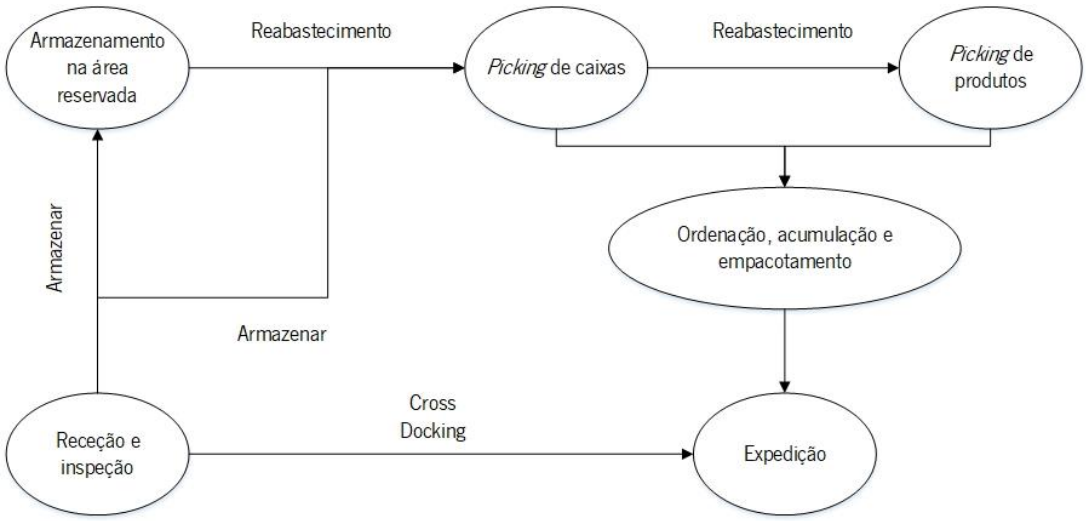

Figura 4 - Operações de armazém. Imagem adaptada de Gong et al. (2009).

O primeiro processo que ocorre num armazém é a receção dos diferentes produtos que chegam via transporte interno ou externo, como camiões. Nesta etapa pode existir um controlo e inspeção dos produtos, ou até mesmo alguma transformação (Rouwenhorst et al., 2000). A receção dos produtos é responsável por 10% dos custos das operações de armazém (Gong et al., 2009). Depois da fase de receção, os itens são armazenados nos locais destinados, que podem ser atribuídos de forma aleatória ou segundo alguma política de armazenamento (Gu et al., 2010). Esta etapa representa 15% dos custos totais das operações de armazém (Gong et al., 2009). O terceiro processo que ocorre num armazém é a resposta a uma encomenda, em que os produtos são recuperados do local onde estão armazenados e entregues na zona de saída. A sequência de recolha dos produtos pode ser feita de forma aleatória ou ordenada segundo alguma política implementada. Esta operação representa a maior fração dos custos operacionais, 55%, sendo por isso determinante (Gong et al., 2009). Após a recolha, os artigos podem ser verificados, transformados ou embalados e, posteriormente, despachados para algum meio de transporte como, camiões, comboios ou outro transportador. Esta é a última etapa dos produtos num armazém e representa 20% dos custos operacionais (Gong et al., 2009; Rouwenhorst et al., 2000).

2.3.3 Recursos de um Armazém

No interior de um armazém podem ser identificados vários recursos. No entender de Rouwenhorst et al. (2000), os recursos que mais se destacam são:

- Unidade de armazenamento: local onde os produtos podem ser armazenados, tais como, paletes, caixas de cartão, caixas de plástico, entre outras.
- Sistema de armazenamento: local onde as unidades de armazenamento são colocadas, como por exemplo, prateleiras e estantes.
- Sistema de recolha dos itens: o *picking* e a movimentação dos artigos podem ser feitos manualmente ou por mecanismos automatizados.
- Sistema de computador: software que permite o controlo de todos os itens e processos do armazém.
- Operários: grande parte dos armazéns são largamente dependentes do trabalho humano.

2.3.4 Políticas das Operações de Armazém

Grande parte dos custos logísticos da CA estão associados às operações que ocorrem nos armazéns (Rouwenhorst et al., 2000). Por esta razão, uma boa definição de todos os processos, desde a fase de design até às operações, é fundamental para que todos os custos de movimentação e armazenamento de produtos sejam minimizados (Cormier & Gunn, 1992). Em seguida serão descritas algumas políticas de armazenamento e de *picking*.

Políticas de Armazenamento

Os principais objetivos das políticas de armazenamento prendem-se, sobretudo, com a flexibilidade do armazém, a otimização do espaço utilizado, a facilidade de circulação de pessoas e máquinas, a simplificação dos fluxos de materiais, as condições de segurança dos produtos e a otimização da utilização dos recursos (Dolgui & Proth, 2010).

As principais políticas de armazenamento são:

- Armazenamento Aleatório

Neste tipo de armazenamento, o operador, ou sistema automático, escolhe aleatoriamente o local onde vai colocar o produto recebido (Rouwenhorst et al., 2000). Todas as *racks*, ou zonas de

armazenamento, têm a mesma probabilidade de receber o item. O armazenamento aleatório resulta numa diminuição do espaço necessário, contudo, aumenta a distância a percorrer nas operações de *picking* (Gu et al., 2010). Em sistemas manuais, esta política resulta numa estratégia de ocupar o local de armazenamento livre mais próximo, em que os operários inserem os artigos no primeiro espaço vazio que encontram. Por esta razão, os locais próximos das zonas de entrada ficam completamente preenchidos, enquanto que, as *racks* mais afastadas vão ficando gradualmente mais vazias (de Koster et al., 2007).

- Armazenamento Pré-Definido

Ao contrário da política anterior, no armazenamento pré-definido os produtos já têm um local prescrito no qual devem ser armazenados (Rouwenhorst et al., 2000). A implementação desta política resulta numa diminuição da distância percorrida, quando comparada com o armazenamento aleatório (Gu et al., 2010), e permite que os operários se familiarizem com a localização dos diferentes produtos. No entanto, a necessidade desta política de garantir, de forma permanente, o espaço suficiente para o stock máximo possível de qualquer artigo, leva a que existam zonas reservadas, incluindo as de melhor acesso, mesmo que não exista essa referência em *stock* (de Koster et al., 2007).

- Armazenamento Baseado em Classes

O armazenamento baseado em classes atribui zonas específicas do armazém a determinados grupos de produtos (Rouwenhorst et al., 2000). Uma forma clássica de divisão dos artigos é o método de Pareto, que se baseia na ideia de categorização dos produtos em três classes diferentes, estratégia ABC. Segundo este método, aproximadamente 20% dos produtos contribuem para 80% da rotação total, logo são os de maior importância e podem ser classificados como produtos da classe A. A divisão das restantes classes pode ser realizada seguindo várias estratégias como uma percentagem fixa de produtos ou de rotação em relação ao total. Após a identificação dos artigos pertencentes a cada uma das classes, A, B e C, é lhes atribuída uma zona de armazenamento específica (de Koster et al., 2007). As zonas de melhor acessibilidade são preenchidas com os produtos das classes mais importantes. Por norma, o preenchimento das *racks* dentro de cada região definida é feito de forma aleatória.

- Armazenamento Baseado na Rotatividade

Nesta estratégia os produtos são armazenados de acordo com a sua rotatividade. A principal diferença para a política anterior, reside no facto dos produtos não serem divididos em classes, mas sim analisados individualmente. Um dos métodos mais utilizados para definir a ordem de alocação dos

produtos é o COI (*cube-per-order índex*), que é definido como a razão entre o espaço total necessário para armazenar um produto e o número de movimentações necessárias para satisfazer a procura. Os produtos com menor COI são armazenados nas zonas de melhor acessibilidade. O autor de Koster et al. (2007) verificou, através de simulação, que o armazenamento baseado em rotatividade origina uma menor distância percorrida nas tarefas de *picking*, quando comparada com a política baseada em classes. No entanto, a principal desvantagem desta política é que, a constante variação da procura dos produtos obriga a frequentes mudanças na ordem e distribuição dos artigos no armazém.

Para além das quatro políticas apresentadas acima, a literatura revela ainda outras menos utilizadas, tais como:

- Armazenamento baseado em famílias, em que se estabelecem relações de procura entre os produtos (de Koster et al., 2007). As famílias podem ser definidas de acordo com as similaridades dos produtos ou agrupando artigos que frequentemente são requeridos em simultâneo (Rouwenhorst et al., 2000).
- Armazenamento baseado em DOS (*Duration-of-Stay*), que é uma extensão do armazenamento baseado em classes, e analisa o tempo de permanência dos produtos no sistema. Os resultados dos estudos realizados demonstraram que esta política reduz significativamente a distância percorrida nas diferentes movimentações (Goetschalckx & Ratliff, 1990). No entanto, esta política é difícil de implementar e, por essa razão, raramente utilizada (Gu et al., 2010).
- Combinação de várias políticas, de Koster et al. (2007) compilou alguns estudos em que se verificou que a combinação de diferentes políticas pode permitir unir as vantagens e reduzir as desvantagens, em comparação com a implementação individual de cada estratégia.

Políticas de *Picking*

As operações de *picking* por sua vez representam aproximadamente 55% dos custos associados às operações de armazenamento (Gong et al., 2009). Por esta razão, é determinante implementar uma política de *picking* que otimize a eficiência dos recursos e, consequentemente, diminua os custos associados. As estratégias de *picking* têm como objetivo primordial, evidenciar uma alternativa vantajosa, em relação à recolha de um artigo único por viagem (de Koster et al., 2007).

- *Picking* por Zonas

Nesta política, a área total de armazenamento pode ser divida em várias zonas de *picking* (Rouwenhorst et al., 2000). Na maioria das situações, cada operário é responsável por recolher dentro da área de *picking* que lhe é atribuída, o que diminui o congestionamento e permite a familiarização dos funcionários com a localização de cada produto. A grande desvantagem desta estratégia é que as encomendas são divididas e recolhidas por diferentes operários (de Koster et al., 2007). Para lidar com esta desvantagem são possíveis duas alternativas, zonas sequenciais ou zonas paralelas (Rouwenhorst et al., 2000). Na estratégia de zonas sequenciais, cada operário recolhe e despacha os produtos da sua zona, se solicitados, e passa a encomenda para a região seguinte. Este processo sequencial é repetido até que todos os produtos requeridos estejam na encomenda. Na segunda estratégia, de zonas paralelas, todos os operários recolhem em simultâneo na sua zona, os artigos de uma só encomenda (de Koster et al., 2007).

- *Picking* por Lotes

Ao contrário da estratégia anterior, no *picking* por lotes cada operário é responsável por várias encomendas em simultâneo e pode circular por todo o armazém (Gu et al., 2010). Esta estratégia é útil em situações em que as encomendas são de quantidade reduzida. Uma vez mais, existem duas estratégias principais que podem ser aplicadas: lotes baseados em proximidade ou lotes baseados numa janela de tempo. Na estratégia de recolha de lotes por proximidade, as encomendas são agrupadas atendendo à proximidade dos produtos que requerem. A dificuldade desta alternativa é determinar qual a melhor sequência de recolha dos diferentes itens de modo a minimizar o tempo de *picking* total. Na segunda alternativa todas as encomendas que chegam dentro de uma janela de tempo, são agrupadas e atribuídas a um só operário. Por sua vez, este operário pode ordenar os itens necessários para as diferentes encomendas antes ou depois da recolha (de Koster et al., 2007).

- *Picking* por Rotas

No *picking* por rotas é definido previamente um percurso e uma sequência de recolha dos produtos (Rouwenhorst et al., 2000). O objetivo desta política é otimizar a ordem pela qual os artigos são recolhidos durante a rota definida. O *picking* por rotas é um caso especial do Problema do Caixeiro Viajante, em que um vendedor, inicialmente na sua cidade, tem que visitar um conjunto variado de cidades uma única vez e retornar a casa. Através do conhecimento de todas as distâncias a percorrer, o objetivo é minimizar

a distância total percorrida pelo vendedor. Várias heurísticas foram desenvolvidas, baseadas no Problema do Caixeiro Viajante, para resolver o problema do *picking* por rotas (de Koster et al., 2007).

Na literatura existem ainda outras políticas, como a seleção do ponto de permanência dos equipamentos em sistemas automatizados, em situações de inatividade (Rouwenhorst et al., 2000). Em alguns sistemas, as políticas de recolha dos produtos são combinadas de modo a otimizar o funcionamento das operações. Por norma, quando as políticas de *picking* por zonas ou *picking* por lotes são implementadas, é necessário um esforço suplementar para acumular e ordenar as encomendas para serem despachadas (de Koster et al., 2007). As encomendas podem ser ordenadas de acordo com uma sequência específica e, posteriormente, encaminhadas para a zona de *output* destinada (Gu et al., 2010). Com a implementação de sistemas automatizados, e de acordo com os princípios da Indústria 4.0, a filosofia JIT/JIS, em que os produtos são em entregues no tempo exato pela sequência certa, é um requisito que os sistemas de armazenamento terão de cumprir.

Ao longo das últimas décadas várias soluções e configurações para os sistemas de armazenamento foram desenvolvidas. Algumas das mais relevantes serão apresentadas nas secções seguintes.

2.4 Armazéns Automáticos

Os armazéns automáticos são sistemas cujas operações são realizadas de forma mais ou menos automatizada por máquinas autónomas e sistemas inteligentes. Uma eficiente utilização destes sistemas resulta em diversas vantagens, tais como: controlo de inventário, otimização do tempo e do espaço, diminuição do risco de erro humano, entre outros (Manzini, 2012). Com o surgimento da Indústria 4.0, as empresas têm procurado implementar sistemas de armazenamento automáticos cujo funcionamento seja regido pelos princípios desta nova filosofia. Nesta secção serão abordados alguns dos sistemas mais estudados e referenciados na literatura.

2.4.1 Armazéns AS/RS

Os armazéns AS/RS são uma das mais importantes ferramentas de movimentação, armazenamento e controlo, desenvolvidas e aplicadas nas CA (Hur, Lee, Lim, & Lee, 2004). Este sistema tem sido largamente utilizado em ambientes de produção e distribuição, desde a sua introdução nos anos cinquenta (Roodbergen & Vis, 2009). Os AS/RS podem desempenhar uma função essencial na

indústria moderna para o armazenamento do WIP (*Work-in-Process*), garantindo o controlo do inventário e a otimização da eficiência dos recursos, do tempo e do espaço (Manzini, 2012).

A grande capacidade destes sistemas levou a que muitas variações e configurações fossem desenvolvidas. A Figura 5 mostra as diferentes possibilidades e as classificações dos armazéns AS/RS, de acordo com as funções que podem ter. Os guindastes, também conhecidos como máquinas S/R (*Storage and Retrieval)*, são equipamentos completamente automatizados, que armazenam e recolhem os produtos. As máquinas S/R podem-se movimentar exclusivamente num corredor ou, noutros sistemas, permutar entre corredores. O transporte realizado, tal como a Figura 5 mostra, pode ser de um só produto, ou de dois artigos em simultâneo. Quanto ao processo de *picking*, pode ser necessário a intervenção humana a bordo do guindaste ou pode ser realizado de forma completamente automatizada. O sistema mais comum utiliza uma máquina S/R automatizada que recebe ou entrega os produtos a um operário nas zonas de I/O (*Input/ Output*). Os armazéns AS/RS podem transportar e armazenar os produtos em paletes ou em caixas.

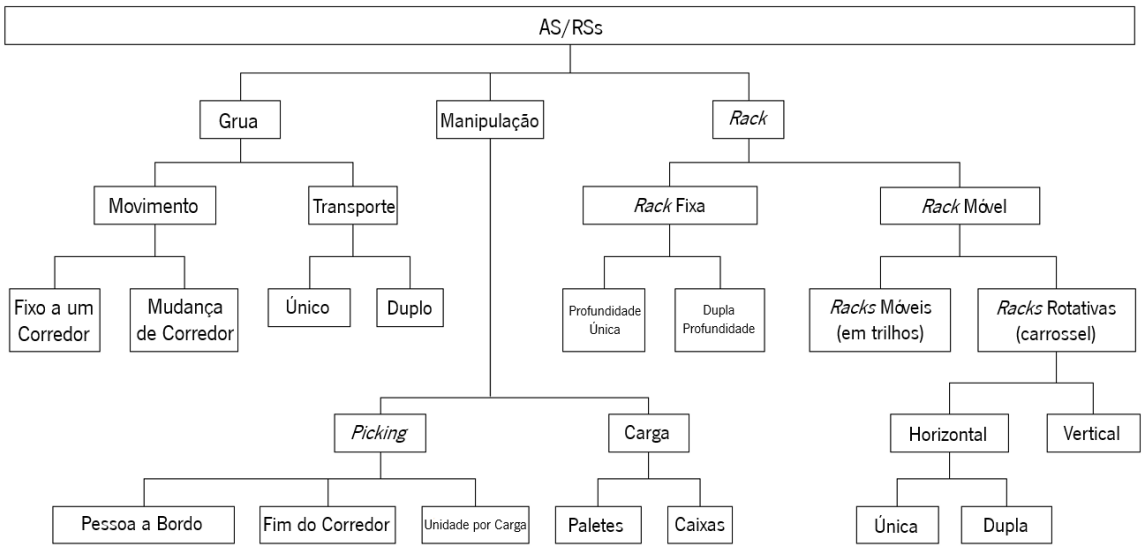

Figura 5 - Sistemas AS/RS. Imagem adaptada de Roodbergen & Vis (2009).

Nas zonas de armazenamento, os AS/RS podem ser constituídos por *racks* fixas ou por *racks* móveis. As *racks* fixas podem armazenar uma só palete ou podem ter dupla profundidade. Já as *racks* móveis podem mover-se em trilhos ou constituir um sistema rotativo denominado carrossel que será abordado em maior detalhe na subsecção 2.4.2 (Roodbergen & Vis, 2009).

Da revisão da literatura, foi possível constatar que um elevado número de pesquisas têm sido feitas ao longo dos últimos anos, no sentido de avaliar e otimizar a estrutura física, as características

operacionais e as políticas de controlo dos armazéns AS/RS (Hur et al., 2004; Manzini, 2012; Roodbergen & Vis, 2009). Na fase de design os aspetos mais importantes a considerar devem ser: a finalidade do armazém, o tipo de produtos a armazenar, a localização das zonas de I/O, as operações a realizar, entre outros (Roodbergen & Vis, 2009). Por outro lado, várias políticas de controlo dos AS/RS, foram avaliadas com base no tempo de movimentação do sistema. Manzini (2012) afirmou que, apesar de várias políticas de controlo da posição de inatividade da máquina S/R terem sido sugeridas, em sistemas com elevada taxa de utilização, a política implementada não aparenta ter grande importância. O autor defende ainda que algumas estratégias de realocação e rearranjo do sistema de armazenamento, em situações de inatividade, melhoram significativamente a performance. No entanto, estas políticas dependem da configuração e operacionalidade do armazém em estudo pelo que, novas pesquisas, comparando diferentes estratégias e configurações são necessárias (Manzini, 2012; Roodbergen & Vis, 2009).

2.4.2 Armazéns em Carrossel

Os sistemas de armazenamento em carrossel são armazéns AS/RS desenhados especialmente para produtos de pequena e média dimensão (Hwang & Ha, 1994). Estes sistemas têm sido largamente usados, em especial em operações de recolha manual de itens (Bengü, 1995). Os armazéns em carrossel mais comuns são compostos por uma serie de caixas, unidas num circuito fechado, montadas sobre uma pista oval. Sempre que um item é solicitado, o carrossel, acionado através de um sistema de controlo, roda até que a caixa onde o item está inserido fique disponível na zona de I/O (Vickson & Fujimoto, 1996). A versatilidade destes sistemas permite que várias configurações, tamanhos e tipos sejam possíveis. Estes sistemas de armazenamento podem ter movimento horizontal ou vertical e a sua rotação pode ser uni ou bidirecional. A grande vantagem dos armazéns em carrossel é que, ao invés do operário despender tempo na viagem até ao item pretendido, é o próprio produto que vem ao encontro do operário (Litvak & Vlasiou, 2010). O interesse por estes sistemas tem levado a que ao longo das últimas décadas diversas investigações tenham sido publicadas, tal como Litvak & Vlasiou (2010) mostraram. O principal problema, e o mais estudado, é a localização ótima dos produtos, uma vez que condiciona o funcionamento de todo o sistema (Bengü, 1995; Litvak & Vlasiou, 2010; Vickson & Fujimoto, 1996; Vol, 1994). Este interesse da indústria e da comunidade científica nos sistemas de armazenamento em carrossel têm levado a que diversos sistemas, baseados nos seus princípios, tenham sido desenvolvidos.

2.5 Armazéns de Alta Densidade

A utilização eficiente do espaço é uma preocupação de todas as empresas. Ao longo dos anos, os gestores e engenheiros dos armazéns e centros de distribuição procuraram encontrar formas de aumentar a densidade de armazenamento. Os sistemas com *racks* de dupla-profundidade, em que uma palete é colocada atrás de outra, foram os primeiros a surgir. Posteriormente, o nível de profundidade e, consequentemente, de densidade, foi aumentando sucessivamente, permitindo que os armazéns acumulassem mais paletes por unidade de área (Gue, 2006). Os desafios da Indústria 4.0 na CA, como a necessidade de sistemas JIT/JIS, a flexibilidade de adaptação às tendências de mercado, o aumento da eficiência dos recursos, como o espaço de armazenamento, entre outros, elevou o interesse da indústria nestes sistemas de alta densidade (Cardin, Castagna, Sari, & Meghelli, 2012). Na última década, várias investigações foram conduzidas, com o intuito de desenvolver e implementar armazéns de alta densidade em conjunto com as funcionalidades dos AS/RS. Um dos sistemas mais estudados e citados na literatura são os armazéns de alta densidade AS/RS com *flow-racks* (Cardin et al., 2012; Ghomri & Sari, 2015; Lehnfeld & Knust, 2014; Sari, Grasman, & Ghouali, 2007; Yu & De Koster, 2012).

As *flows-racks* são *racks* com elevada profundidade constituídas, na maioria dos casos, por caixas sustentadas em transportadores, ou suportes, que se movimentam por ação da gravidade (Cardin et al., 2012). Os sistemas com *flow-racks*, tal como os armazéns de alta densidade, apresentam vantagens ao nível da eficiência de espaço e do tempo de armazenamento. No entanto, a necessidade de, em algumas circunstâncias, inserir produtos de referências diferentes na mesma *flow-rack*, leva a que o acesso a determinados artigos esteja bloqueado pelos que se encontram à sua frente (Gue, 2006). Desta forma, para se alcançar um produto em posições para além da primeira, é necessário movimentar e voltar a armazenar todos os artigos que o antecedem. Uma vez que, o mesmo artigo pode estar em diversas *flow-racks* e com acessibilidade diferente, surge assim um problema de seleção do item que irá abandonar o sistema de armazenamento (Lehnfeld & Knust, 2014). O produto selecionado deve minimizar o tempo e as movimentações, cumprindo, sempre que possível, a regra FIFO. A política FIFO é das mais utilizadas no contexto real e procura garantir que o primeiro produto de uma determinada referência a chegar ao sistema é o primeiro a sair. Assim, evita-se que existam produtos obsoletos, uma vez que, o produto despachado é sempre o mais antigo (Yu & De Koster, 2012). Para resolver estes problemas, vários autores investigaram a influência que o design e as políticas de armazenamento e de *picking* poderiam ter no funcionamento dos sistemas de alta densidade. Apresenta-se de seguida o estado de arte dos armazéns AS/RS de alta densidade, com especial enfoque para os que contêm *flow-racks*.

Gue (2006) desenvolveu um algoritmo capaz de determinar a configuração dos armazéns de alta densidade, que minimiza o número de interferências no armazenamento dos itens. Desta forma, é possível reduzir o tempo de recolha dos artigos nas operações de *picking*.

Sari et al. (2007) elaboraram um modelo para medir o tempo de ciclo de um AS/RS com *flow-racks*, baseado nas políticas de armazenamento e *picking* aleatório, para situações em que existe mistura de referências na mesma *rack*. Este estudo teve como objetivo fornecer uma base de comparação para futuras investigações que utilizem políticas de armazenamento e *picking* mais elaboradas.

No mesmo ano, Gue & Kim (2007) introduziram uma técnica original para resolver o problema de movimentação de paletes, em armazéns de alta densidade. O método proposto baseava-se no puzzle de 15 peças com um tabuleiro em grelha de 4x4, cujo objetivo é conseguir resolver o puzzle qualquer que seja a configuração inicial. As peças representam as paletes e o tabuleiro as diferentes *racks*. Para resolver este problema, os autores desenvolveram um algoritmo que se revelou eficaz na procura da melhor sequência que organiza as paletes com a distribuição pretendida e que minimiza o número total de movimentações.

Yu & de Koster (2009) procuraram encontrar as fronteiras para as zonas de armazenamento de duas famílias diferentes de produtos, uma de produtos de elevada rotação e outra com os de baixa rotação. O estudo realizado comprovou que a implementação desta política reduz o tempo de viagem do sistema, em comparação com a política de armazenamento aleatório.

Três anos mais tarde, os mesmos autores publicaram uma investigação em que avaliaram, através de vários algoritmos, a escolha correta da sequência de recolha de itens num AS/RS 3D, de alta densidade. Tratando-se de um problema NP-difícil e, portanto, difícil de tratar de forma prática, os autores aplicaram quatro heurísticas de sequenciamento: FIFO, vizinho mais próximo, SDC (*Shortest Dual-command Cycle*) e PPR-SL (*Percentage Priority to Retrievals with the Shortest Leg*). Os resultados, conseguidos através de simulação, mostraram que o PPR-SL, a heurística com melhor performance, supera em 20% a segunda melhor heurística (SDC). A heurística do vizinho mais próximo e o cumprimento do FIFO levam a piores resultados no tempo total despendido para completar uma encomenda (Yu & De Koster, 2012).

No mesmo ano, Cardin et al. (2012) apresentaram um novo método de armazenar e recolher os produtos em armazéns AS/RS com *flow-racks* denominado *In-Deep Class Storage*. Este método baseia-se no princípio de que é mais eficiente alocar às *racks* mais próximas da saída os produtos com maior rotação. Os autores demonstraram, através da simulação e da aplicação de dois algoritmos, que reservar

as primeiras camadas de cada *rack* para os produtos de maior procura, resulta numa diminuição do atraso médio da recolha dos produtos.

Lehnfeld & Knust (2014) publicaram uma revisão da literatura onde analisaram os problemas de carregamento dos produtos que chegam ao sistema, de descarregamento dos artigos requeridos numa encomenda, de ordenação e rearranjo dos itens e, por fim, problemas combinados, em que em simultâneo existem itens a entrar e a sair do armazém. Os autores apresentaram um sistema de classificação de forma a compilar os resultados e as soluções encontradas para todas as classes de problemas. De acordo com Lehnfeld & Knust (2014), apesar de diversos algoritmos e soluções já terem sido publicadas, em situações complexas e de requisitos específicos será necessário desenvolver novos métodos e algoritmos.

Gharehgozli, Yu, Zhang, & De Koster (2014) criaram e aplicaram um algoritmo de tempo polinomial, a diversas configurações de armazéns AS/RS de alta densidade, com o objetivo de minimizar o tempo total de viagem da máquina S/R. Os autores formularam o problema, que é um caso especial do Caixeiro Viajante, NP-difícil, e aplicaram o algoritmo desenvolvido. O algoritmo melhorou em 30% e 15% o tempo total de ciclo, em comparação com o sistema FIFO e vizinho mais próximo, respetivamente.

No ano seguinte, Zaerpour, Yu, & De Koster (2015) estudaram o armazenamento de produtos frescos num centro de distribuição. O principal problema encontrado foi a necessidade constante de reorganizar os produtos, devido à incorreta sequência de armazenamento. Para resolver este desafio, os autores propuseram um modelo matemático para uma política de armazenamento partilhado, que minimiza o tempo total de *picking*. Posteriormente, foi proposta uma heurística capaz de resolver, de forma efetiva e eficiente, problemas de dimensão real. Os autores concluíram que, apesar das vantagens da política de armazenamento dedicado, em que uma *rack* apenas contém produtos de uma só referência, a estratégia partilhada, onde há mistura de referências na mesma *rack*, supera a primeira no tempo de resposta e na utilização do espaço.

Ghomri & Sari (2015) modelaram matematicamente o tempo médio de viagem da máquina S/R em funções de recolha, num sistema de armazenamento aleatório. O estudo foi desenvolvido num armazém AS/RS com *flow-racks*, com uma configuração específica. Segundo os autores, o modelo desenvolvido e validado através de simulação, é útil, mas válido apenas para armazéns do mundo real com configurações semelhantes ao estudado.

Metahri & Hachemi (2018) desenvolveram um modelo analítico para estudar o tempo de viagem no processo de recolha de produtos armazenados num AS/RS com *flow-racks* por gravidade, seguindo a política de armazenamento pré-definido. Este novo tipo de *flow-racks* tem inúmeras vantagens em

comparação com as convencionais, tais como, maior taxa de transferência, configuração mais flexível e menor investimento inicial. O modelo criado foi avaliado através de simulação discreta e revelou-se eficaz no tratamento deste problema.

Wang, Yang, & Li (2019) analisaram várias configurações de um sistema de armazenamento de *racks* movimentados por robot. Com a utilização dos modelos criados para determinar o tempo de ciclo do sistema, os autores conseguiram perceber os melhores parâmetros na configuração deste armazém, como a profundidade das *racks*. Apesar do enorme interesse da indústria e da comunidade científica, os autores defendem que cada armazém, devido às suas especificidades e às características dos seus produtos, necessita de ser estudado individualmente no sentido de determinar o melhor design e as melhores políticas operacionais.

3. APRESENTAÇÃO DO PROBLEMA REAL

Este capítulo contém, numa primeira parte, uma descrição da empresa onde o Shopstocker foi desenvolvido. Segue-se a apresentação do sistema de armazenamento em estudo, é descrito o seu funcionamento e as suas especificações. Na última parte deste capítulo são identificados os problemas reais existentes no Shopstocker implementado.

3.1 A empresa: Pinto Brasil S.A.

Fundada em 1991, a Pinto Brasil S.A. (PB) tem dedicado a sua atividade ao desenvolvimento de soluções que satisfaçam as necessidades dos seus clientes (Brasil, 2019). Esta empresa, criada por Manuel Pinto Brasil, a partir de um anexo em S. João da Ponte no distrito de Braga, opera na área da metalomecânica com especial enfoque no setor da indústria automóvel e aeronáutica (Fiel, 2010). A Figura 6 mostra o logótipo da empresa.

Figura 6 - Logótipo da Pinto Brasil S.A.

Ao longo dos seus vinte e nove anos de existência, a PB teve um crescimento sustentado, internacionalizando o seu negócio para países como Roménia, Tunísia, Marrocos, Espanha, México e Alemanha. A empresa expandiu sucessivamente a sua área de produção até atingir os 20000 m^2 em 2017 (Brasil, 2019). Esta capacidade de produção, assim como a dinâmica e eficiência da sua equipa, permite à PB assegurar o desenvolvimento de soluções para dar resposta às diferentes necessidades dos seus clientes. Devido à natureza dos seus compradores, maioritariamente do setor automóvel e aeronáutico, a PB exporta mais de 90% da sua produção. O seu nicho de mercado mais influente é a conceção de linhas de produção e de montagem e o desenvolvimento de aplicações logísticas, tais como sistemas de armazenamento, sistemas de fornecimento sincronizado e soluções integradas de AGV (*Automated Guided Vehicles*). A PB desenvolve soluções específicas e personalizadas, de acordo com as necessidades encontradas, focadas na simplicidade e segurança dos seus trabalhadores, procurando

deste modo aumentar a produtividade e qualidade dos seus produtos e processos. A experiência adquirida através da conceção e implementação de diferentes projetos, envolvendo profissionais com conhecimento acreditado, assegura que o objetivo final é alcançado, permitindo o desenvolvimento de soluções personalizadas.

No ano de 2000, a PB tornou-se uma empresa certificada de acordo com a norma NP EN ISO 9002 e conseguiu em 2003 o certificado da norma NP EN ISO 9001 (Brasil, 2019). O departamento de qualidade da empresa assegura que todos os processos são rigorosamente controlados, desde o produto até ao processo responsável pela sua obtenção. Do mesmo modo, para garantir os padrões de qualidade aos quais a empresa se obriga a cumprir, a PB realiza o processo completo de treino e instalação do produto no cliente com uma rapidez de topo. A qualidade dos seus produtos é um dos pilares desta empresa, que garante o respeito e a satisfação dos seus compradores.

Com o objetivo de aumentar a competitividade da empresa, e de melhorar a qualidade dos seus produtos, a PB continua o seu processo de investigação e inovação em infraestruturas e tecnologia de produção, através do seu departamento de pesquisa e desenvolvimento. A empresa criada por Manuel Pinto Brasil tem como principais metas o desenvolvimento avançado, a robustez financeira, a lucratividade da empresa, o crescimento sustentado, a eficiência operacional e a responsabilidade social, procurando para isso otimizar a qualidade e ter um serviço ao cliente sólido.

Os produtos desenvolvidos pela PB são adaptáveis às necessidades de clientes de diversas áreas. Esta personalização dos equipamentos possibilita que a PB os instale numa fábrica de produção de produtos de qualquer natureza. A PB concebeu soluções para dar resposta às dificuldades encontradas em vários processos logísticos. O seu principal objetivo passa pela otimização dos sistemas de produção e dos processos logísticos associados, desde linhas de produção, linhas de montagem, transporte e armazenamento de produtos e componentes (Brasil, 2019). A grande capacidade de resposta desta empresa às diferentes necessidades de cada cliente, permite que a PB aceite personalizar e implementar os seus sistemas e produtos em qualquer indústria. Esta capacidade é uma grande vantagem competitiva e diferenciadora da PB. É por todas estas razões que a PB é reconhecida como uma das empresas mais prestigiadas da sua área, sendo o fornecedor preferencial de algumas das maiores marcas da indústria automóvel (Brasil, 2019).

3.2 Shopstocker

O Shopstocker é um dos produtos desenvolvidos e instalados pela PB. Este sistema, que combina o armazenamento com o transporte aéreo, pode ser aplicado a componentes de qualquer natureza. Com o desenvolvimento deste armazém, a PB procurou aumentar a eficiência da cadeia de abastecimento onde este sistema é implementado. O Shopstocker reduz até 65% o espaço ocupado no solo, diminui até 75% a necessidade de mão-de-obra em atividades de transporte e movimentação de componentes e, devido às suas características, pode ser adaptado de acordo com as necessidades do cliente. Este sistema pode carregar até 120 quilos por peça, e o seu design pode ser alterado em função das caraterísticas de cada produto. De acordo com as especificações do produto, o armazenamento e o transporte do mesmo podem ser feitos horizontalmente ou verticalmente. A PB concebeu o Shopstocker de forma a reduzir o tempo despendido pelos operadores a executar as tarefas necessárias e, por essa razão, são necessários apenas alguns segundos para carregar ou descarregar um produto do respetivo suporte (Brasil, 2019).

A Figura 7 mostra algumas das configurações possíveis do armazém e do mecanismo de transporte do Shopstocker.

Figura 7 - Armazém automatizado Shopstocker.

3.2.1 Implementação do Shopstocker numa Empresa do Setor Automóvel

A crescente pressão mediática associada à Indústria 4.0, tem levado a que várias empresas procurem soluções alinhadas com os princípios desta filosofia (Hermann et al., 2016). O setor automóvel, em especial, tem procurado de forma intensiva substituir todos os processos realizados por ação humana por mecanismos automatizados (Helper, Martins, & Seamans, 2019). No seguimento destas tendências, e aproveitando uma oportunidade de mercado, a PB desenvolveu um projeto de investigação com o objetivo de conceber e implementar um sistema integrado de transporte e armazenamento automático de para-choques (PC), complementado com um *software* de gestão. O sistema desenvolvido tira proveito do produto descrito acima, o Shopstocker, e foi implementado pela PB num armazém de uma empresa fornecedora de PC automóveis, que será designada neste projeto por CLxPB. Este armazém atua numa fase final da CA, recebendo os PC completamente finalizados e armazenando-os até que a sua movimentação seja solicitada para satisfazer uma determinada encomenda.

3.2.2 Caracterização do Shopstocker

O Shopstocker é um sistema de armazenamento e transporte aéreo automatizado adaptado, no caso em estudo, com suportes de PC. O armazém implementado na CLxPB está dividido em duas plantas, para-choques frontais e traseiros, sendo que, neste projeto de dissertação, optou-se por analisar a planta dos PC frontais.

A planta estudada é constituída por dois andares. No piso inferior desta instalação existe uma zona de *Input*, uma zona de *Output*, uma zona de saída rápida e 43 linhas de armazenamento. Já no piso superior existem 43 linhas, sendo que duas servem para pré-sequenciamento de PC da encomenda seguinte. A movimentação dos suportes entre os dois pisos é realizada com auxílio de quatro elevadores, dois para subida e dois para descida dos suportes. Apesar do elevado nível de automatização, os processos de introdução e extração dos PC nos respetivos suportes, nas zonas de *Input/Output*, são realizados por operários. Após a colocação do PC, o suporte inicia o percurso até à extremidade de entrada da linha destinada, através de um sistema de tração do tipo carrossel, evidenciado a preto na Figura 8. O sistema de tração incorpora vários circuitos independentes de cordas tracionadas por um motor e por escovas. Estas escovas distam 1,5 metros entre si e são responsáveis pela movimentação dos suportes. Apesar da velocidade de movimentação da corda ser de 300 milímetros por segundo, a deslocação dos suportes é inferior, uma vez que está dependente da fixação do suporte por uma das escovas existentes no circuito. Os suportes, com ou sem para-choques, ficam armazenados desde que

são inseridos nas linhas, até que seja solicitada a sua movimentação. Devido ao *design* dos PC o seu transporte e armazenamento é realizado na vertical.

A Figura 8 ilustra de forma simplificada o funcionamento do Shopstocker, evidenciando os seus fluxos.

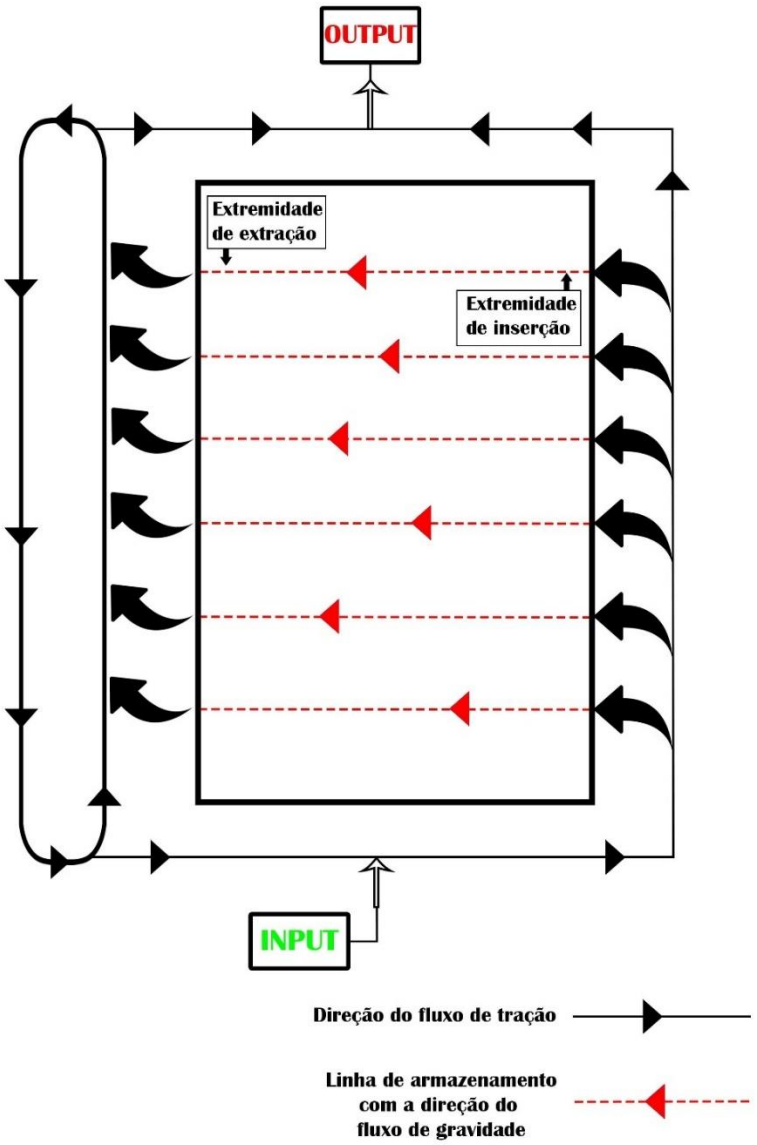

Figura 8 - Esquema simplificado do funcionamento do Shopstocker.

O sistema implementado na CLxPB tem uma capacidade de armazenamento até 2130 para-choques e mais de 100 referências diferentes, evitando uma possível quebra de fornecimento durante três dias. Os PC são armazenados em linhas concebidas para que, através da gravidade, os suportes se movimentem da extremidade de inserção até à extremidade de saída. Este fluxo de gravidade está representado a vermelho na Figura 8. Estas linhas têm capacidade para armazenar 25 suportes

cada. O Shopstocker está atualmente em funcionamento, laborando diariamente durante dois turnos de 8 horas cada.

O processo de extração dos suportes, alvo do estudo neste projeto, tem quatro tempos associados: tempo de extração da posição, tempo de extração da linha, tempo de elevador e tempo total de extração.

Para que um PC seja extraído da sua posição de armazenamento através do fluxo de gravidade, são necessários 8 segundos por posição. Assim, a extração de um PC da primeira posição (posição mais próxima da extremidade de extração) demorará 8 segundos, enquanto que a saída da posição 20 custará ao sistema 160 segundos (8 x 20). A este tempo convencionou-se chamar de tempo de extração da posição.

O fluxo de tração do Shopstocker movimenta os PC a uma velocidade de 150 milímetros por segundo. Uma vez que as linhas de armazenamento distam 0,75 metros entre si, o tempo de percurso entre duas linhas consecutivas é de 5 segundos. Desta forma, retirar um PC da linha 40 obriga a gastar 200 segundos (5 x 40) até este atingir a linha 1 (linha de referência). A este tempo associado ao fluxo de tração denominou-se de tempo de extração da linha.

O terceiro tempo, tempo de elevador, está associado ao transporte dos PC do piso superior para o piso inferior. Assim, sempre que um PC do piso superior é selecionado, o sistema demora 20 segundos a transportá-lo para o piso inferior. Por esta razão, às posições e linhas de armazenamento do piso superior é acrescentado o valor constante de 20 segundos em comparação com os tempos do piso inferior. A este tempo constante convencionou-se chamar de tempo de elevador.

Por fim, o último tempo referido neste projeto é o tempo total de extração. Este tempo corresponde ao somatório do tempo de extração do PC da linha e da posição de armazenamento, acrescido ou não, do tempo de elevador.

3.2.3 Política de Armazenamento dos Para-Choques

A grande diversidade de PC que o Shopstocker implementado na CLxPB armazena advém da variedade de especificações e personalizações que podem ser feitas e pedidas pelo cliente, tais como, cor, forma, modelo, entre outras. Apesar da variedade de opções, os para-choques estão divididos em três classes de acordo com a sua procura: *high runners*, *medium runners* e *low runners*. A análise e atualização das referências pertencentes a cada classe é realizada mensalmente de forma a assegurar que a disposição dos PC no armazém é consentânea com a procura efetiva dos mesmos.

Os *high runners* correspondem aos PC com maior procura por parte do cliente. O volume de PC em armazenamento destas referências é maior e o seu tempo de permanência no sistema é menor quando comparado com as restantes classes. Estes PC são armazenados exclusivamente no primeiro piso nas linhas destinadas a armazenar os *high runners*, nas quais não há mistura de referências.

Já os *medium runners* possuem uma procura superior aos *low runners*, mas inferior aos *high runners*. As linhas destinadas aos PC desta classe podem conter até três referências diferentes.

Os *low runners*, por sua vez, são os PC com menor procura pelo que, a sua quantidade em armazenamento é menor. Nas linhas destinadas aos *low runners* não há número limite de referências diferentes, pelo que, num caso extremo, será possível armazenar vinte e cinco referências diferentes na mesma linha.

Resumindo, existem três tipos de linhas de armazenamento diferentes: linhas dedicadas exclusivamente a *high runners* (uma referência por linha), linhas mistas reservadas a *medium runners* (até três referências por linha) e linhas caóticas destinadas a *low runners* (sem limite de referências por linha).

3.2.4 Satisfação de uma Encomenda

Sempre que uma encomenda de PC é feita à CLxPB, o armazém recebe a informação das referências e da sequência pretendida pelo cliente, com duas horas de antecedência. Uma encomenda de PC é caracterizada por ser uma lista ordenada de referências num tamanho variável que em média se considera ser 100 PC em cada fornecimento. A sequência de PC assume-se como um fator importante, uma vez que corresponde à ordem de produção dos veículos no cliente da CLxPB, não podendo por isso ser desrespeitada. Os para-choques selecionados pela CLxPB para satisfazer a encomenda são movimentados para a zona de saída pela sequência de referências pretendida pelo cliente, chegando à zona de *output* numa cadência desejável de 80 em 80 segundos. Este *takt time* é também o tempo necessário para remover um para-choques do suporte e o colocar num camião. Cada camião carrega uma única encomenda de cada vez, e tem capacidade para transportar a totalidade dos PC solicitados na encomenda que a CLxPB recebe.

3.2.5 Funcionamento do Shopstocker

Dentro das instalações do Shopstocker existem cinco tipos de movimentações: requisição de suporte vazio, armazenamento de para-choques, armazenamento de suportes vazios, reorganização dos suportes no armazém e saída dos para-choques. Estes processos são descritos detalhadamente de seguida.

Requisição de Suporte Vazio

Sempre que é necessário introduzir um para-choques no Shopstocker, o sistema solicita a movimentação de um suporte vazio para a zona *input*. A Figura 9 ilustra o fluxograma com as diversas decisões que o sistema tem de tomar, dependendo das condições existentes.

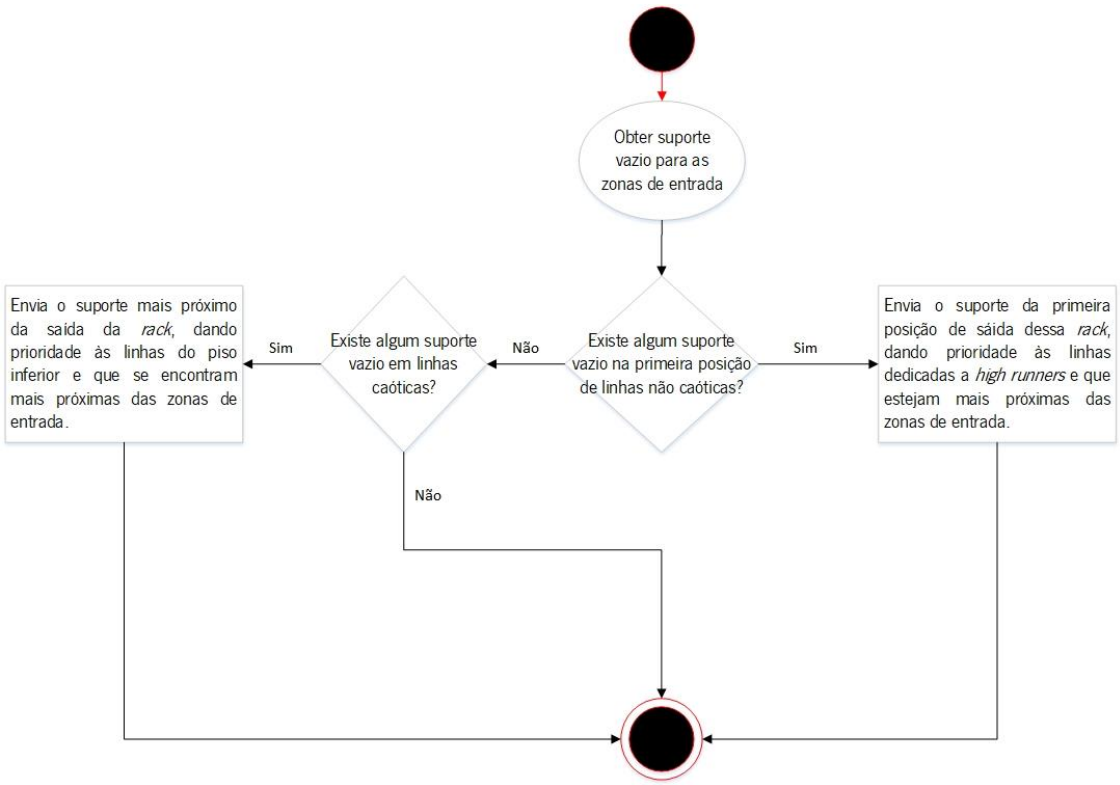

Figura 9 - Fluxograma da requisição de um suporte vazio.

Quando um suporte vazio é solicitado, a primeira verificação que o sistema executa é se existem suportes vazios na primeira posição de saída em linhas não caóticas. Caso a condição se verifique, um suporte vazio é enviado para a zona de entrada, dando prioridade às linhas dedicadas a *high runners* e, dentro das alternativas, a que se encontre mais próxima da zona de *input*. Por outro lado, como o

fluxograma da Figura 9 demonstra, caso não existam suportes vazios em linhas não caóticas, o sistema procede à segunda verificação, isto é, afere se existem vazios em linhas caóticas. Se se confirmar esta segunda condição, é então escolhido o suporte vazio que esteja na posição mais próxima da saída da linha, dando prioridade às linhas do piso inferior que estejam mais próximas da zona de *input*. Em contrapartida, se também não existirem suportes vazios em linhas caóticas, nenhum suporte é enviado para a zona de *input*.

Armazenamento de Para-choques

O processo de armazenamento de um para-choques é bem mais complexo que o anterior, e procura obedecer às premissas definidas na subsecção 3.2.3. Desta forma, após a colocação do PC no suporte na zona de entrada, o sistema verifica se existem linhas dedicadas à referência do modelo inserido que tenham espaço livre para receber o novo produto. Caso esta situação não se verifique, o sistema do Shopstocker averigua uma serie de condições de modo a armazenar o PC inserido na posição mais conveniente de modo a que seja cumprida a política de armazenamento definida.

Armazenamento de Suportes Vazios

Após a entrega do para-choques na zona de *output*, o suporte vazio necessita de um novo armazenamento. Para este efeito, o sistema verifica, numa primeira fase, se existe alguma linha caótica com espaço para receber o suporte. No caso desta condição se verificar, o sistema confirma se essa linha se encontra no primeiro piso e, em caso afirmativo, seleciona a mais próxima da zona de *input*. Na alternativa oposta, o sistema opta por enviar o suporte vazio para a linha caótica disponível no piso superior mais próxima da zona de *input*.

Por outro lado, se nenhuma linha caótica tiver disponibilidade para receber o suporte vazio, o sistema averigua se alguma linha dedicada tem essa capacidade. Em caso afirmativo, o sistema verifica se existe essa possibilidade no segundo piso. Por fim, confirmando-se estas três condições, o sistema armazena o suporte vazio na linha dedicada, do segundo piso, de acordo com a seguinte ordem de prioridade: linhas caóticas dedicadas a *low runners*, linhas reservadas a *medium runners* e, por fim, linhas dedicadas a *high runners*. Contudo, se apenas existir capacidade para armazenar no primeiro piso, o sistema seleciona a linha seguindo a mesma ordem definida acima.

Reorganização dos Suportes

Dentro das instalações do Shopstocker existem três reorganizações possíveis, uma para realocar os suportes vazios e duas para realocar para-choques. Estas reorganizações permitem que a distribuição dos PC ao longo do Shopstocker se mantenha o mais próximo possível da situação ideal definida com política de armazenamento implementada. As reorganizações existentes no Shopstocker são as seguintes:

- Realocação de suporte vazio: caso algum suporte vazio esteja na primeira posição de uma linha dedicada a *high runners* pode ser extraído e realocado numa linha caótica.
- Realocação de PC de linhas caóticas em linhas dedicadas: caso um PC esteja na primeira posição de uma linha caótica e a linha dedicada à referência do mesmo tenha disponibilidade para o receber, o PC é extraído e armazenado na linha dedicada.
- Realocação de PC de linhas dedicadas em linhas caóticas: caso um PC se encontre na primeira posição de saída de uma linha dedicada à qual não pertence é extraído e realocado numa linha caótica que tenha disponibilidade para o receber.

Saída dos Para-Choques

Sempre que uma referência é solicitada na zona de *output*, o sistema averigua uma série de condições para selecionar o PC mais adequado a abandonar o sistema. A seleção do PC tem em consideração a linha e a posição de armazenamento na qual o PC se encontra, o tempo de permanência do PC no interior do armazém e o tempo de extração total desse PC. Este processo de seleção procura assegurar que a regra FIFO e a cadência de extração de PC são cumpridas.

3.2.6 Caracterização do Problema

O processo de *picking* no Shopstocker é realizado de forma automatizada seguindo as regras de seleção descritas na subsecção 3.2.5. Sempre que uma sequência de extração chega, normalmente com duas horas de antecedência, os PC necessários para satisfazer a encomenda são selecionados de acordo com a organização do Shopstocker no momento. A maior dificuldade deste processo prende-se com as restrições que o sistema tem de analisar para escolher, de entre todas as opções, o PC ideal. Sempre que não seja possível retirar o PC escolhido da primeira posição de uma linha de

armazenamento, o Shopstocker tem que proceder à movimentação de PC para realocação, que não podem ser selecionados para a encomenda em curso, o que gera perda de eficiência.

A Figura 10 ilustra de forma simplificada uma sequência de extração com 6 posições e uma configuração do Shopstocker com 9 linhas de armazenamento, 3 para cada classe de PC, com 5 posições cada. Cada PC armazenado possui uma cor (representado uma referência) e ostenta o seu valor de FIFO. Quando menor o valor, mais antigo é esse PC dentro da sua referência.

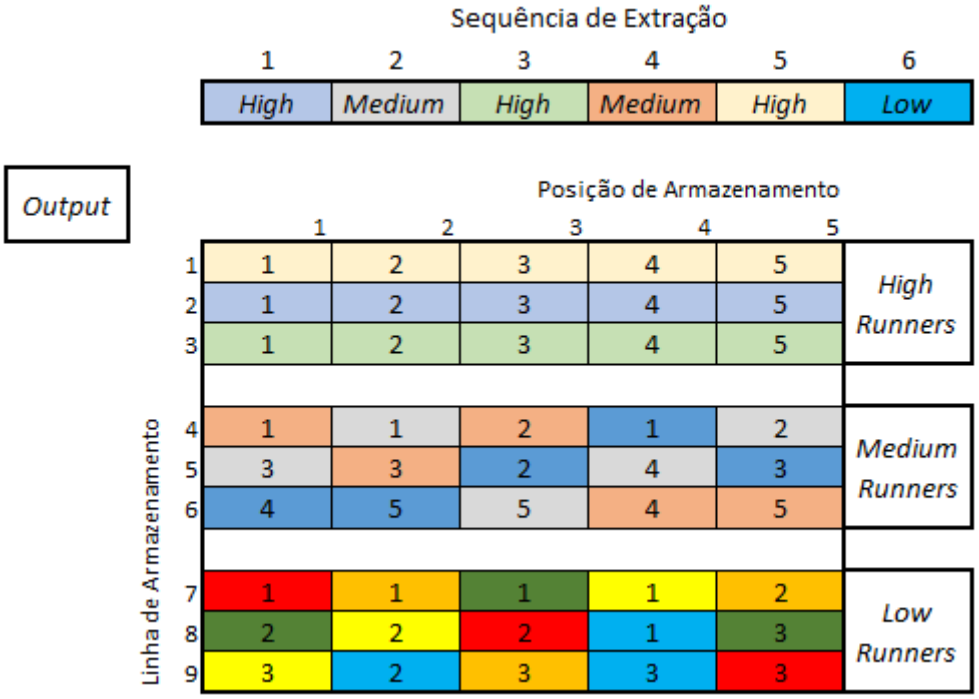

Figura 10 - Representação do problema identificado.

Nas linhas dedicadas a *high runners* em que não há mistura de referências, o processo de escolha é imediato e não prejudica a eficiência do Shopstocker. Na Figura 10 é apresentada uma sequência de extração que requer PC *high runners* nas posições 1 (azul ciano), 3 (verde claro) e 5 (amarelo claro). Uma vez que as linhas onde estes PC estão armazenados são exclusivas e os PC se encontram ordenados pela ordem FIFO, os PC ideais para a extração são os das primeiras posições de armazenamento das linhas 2 (azul ciano), 3 (verde claro) e 1 (amarelo claro), respetivamente. Este processo é por isso imediato e eficiente no Shopstocker.

No entanto, devido à necessidade de armazenar os PC necessários a suprir uma eventual falha no fornecimento durante um período de três dias, a quantidade de *high runners* habitualmente em *stock*, é superior às posições destinadas ao armazenamento dos mesmos. Por esta razão, e por se considerar que as referências desta classe não deveriam ser armazenadas no piso superior, surgiu a necessidade

de criar linhas caóticas, no piso inferior, contendo uma mistura de referências *high runners*. Estas linhas armazenam assim os PC que quando entraram no sistema, não tinham uma linha dedicada disponível. A seleção destes PC, ou a sua realocação em linhas dedicadas, aumenta a entropia do sistema e, consequentemente, prejudica a taxa de extração.

Por sua vez, nas linhas mistas e caóticas com *medium* e *low runners*, respetivamente, a referência pretendida pode estar posicionada em diversos locais ao longo do armazém. O processo de seleção nestas linhas torna-se por isso muito mais complexo. O sistema tem de analisar as diferentes possibilidades de escolha atendendo à regra FIFO, à distância da linha até à zona de saída e, por fim, à posição na linha do PC pretendido, procurando cumprir o *takt time* de 80 segundos.

No exemplo dado na Figura 10, a sequência de extração requer dois PC *Medium Runners* nas posições 2 (cinzento) e 4 (castanho). A seleção dos melhores PC para estas posições é bastante complexa. No caso de se selecionar o PC cinzento da posição 2 da linha 4, primeiro da ordem FIFO, o PC imediatamente à sua frente terá que ser removido e, consequentemente, excluído da encomenda em curso. Assim, não seria possível cumprir o FIFO da cor castanho, uma vez que o PC ideal teria sido removido. Por outro lado, se a opção fosse o PC cinzento na posição 1 da linha 5, nenhum PC extra seria movimentado, porém, não se cumpriria o FIFO desta cor. Este exemplo simples demonstra a complexidade da decisão que o sistema real tem de tomar.

No caso do PC *low runner* azul requerido na posição 6 da sequência de extração da Figura 10 a sua seleção é ainda mais complexa. O Shopstocker tem dificuldade em cumprir a regra FIFO, sem comprometer o *takt time* de 80 segundos e sem gerar demasiada entropia no sistema com a movimentação de PC extra não requeridos na sequência de extração.

Os problemas aqui identificados e caracterizados relativos à seleção dos melhores PC para uma determinada encomenda foram o alvo do estudo realizado neste projeto de dissertação.

4. MODELAÇÃO DO PROBLEMA

Neste capítulo é apresentada a modelação realizada para procurar dar resposta aos problemas identificados no capítulo anterior. Na parte inicial expõe-se a metodologia adotada e é descrita detalhadamente a análise realizada aos dados. Na segunda parte deste capítulo encontra-se exposto o modelo matemático desenvolvido neste projeto, as ferramentas utilizadas e a implementação do modelo matemático em linguagem de programação Python.

4.1 Metodologia Adotada

Após o estudo inicial do problema real de armazenamento no Shopstocker e caracterização das suas restrições funcionais, procurou-se encontrar uma solução através da modelação de um problema de otimização combinatória. Esta área da ciência procura encontrar uma solução ótima para um determinado objetivo cumprindo um conjunto de restrições. O problema de otimização combinatória tem como finalidade maximizar ou minimizar uma função objetivo através do valor atribuído a variáveis de decisão. Estas variáveis são, por sua vez, o alvo da otimização e o seu valor é limitado pelas restrições do problema. Sempre que um problema é passível de ser resolvido com técnicas ou algoritmos exatos, a solução encontrada é ótima, isto é, não existe para aquele problema, uma qualquer outra solução que melhore o valor da função objetivo. Por outro lado, o enredo das situações reais leva a que grande parte dos problemas apresentem uma complexidade intratável computacionalmente em tempo útil. Por esta razão, vários algoritmos foram desenvolvidos baseados em heurísticas que, apesar de não garantirem a solução ótima, fornecem uma boa solução, e em tempo útil, para o problema em análise. Os problemas de otimização podem ser classificados de diversas formas dependendo da relação entre as variáveis de decisão, do valor que estas variáveis podem tomar e da natureza da função objetivo.

Ao observar a situação em estudo neste projeto verifica-se que, numa fase inicial, se assemelha ao problema de afetação que, por sua vez, é um caso particular do problema de transportes. O problema de transportes consiste num conjunto de origens, com uma oferta de produtos limitada, e um conjunto de destinos com uma procura específica por cada produto. O objetivo é transportar a quantidade pretendida pelos destinos, e existente nas origens, minimizando o custo total. Por sua vez, o problema de afetação pode ser visto como um conjunto trabalhos (origem) que necessitam de ser alocados a diversas máquinas (destino). Tal como no problema de transportes, o objetivo é alocar todos os trabalhos às máquinas minimizando o custo total dessa alocação.

De forma similar, o problema de seleção dos melhores PC para uma determinada sequência pode ser visto como um conjunto de origens, PC, com um custo de movimentação associado, e um conjunto de destinos, posições da sequência de extração. O objetivo primordial é assim alocar os melhores PC às diferentes posições da sequência de extração, minimizando o custo associado à sua movimentação. De acordo com estes pressupostos, optou-se por modelar matematicamente o problema em estudo neste projeto através da Programação Linear Inteira Mista (em inglês MILP - *Mixed Integer Linear Programming*), em que algumas variáveis de decisão podem receber valores inteiros (binários) e outras podem receber valores reais. O problema MILP desenvolvido foi definido em linguagem matemática e posteriormente implementado em linguagem de programação Python através do ambiente de desenvolvimento integrado PyCharm.

4.2 Dados do Shopstocker

De modo a desenvolver as instâncias necessárias para o estudo desenvolvido neste projeto foi necessário analisar os dados fornecidos e com eles desenvolver a configuração inicial representativa do estado real do Shopstocker implementado na CLxPB. Para além dessa configuração, e para que fosse possível avaliar o comportamento do modelo durante a sua fase de desenvolvimento foi necessário gerar uma instância de teste com um volume limitado de dados. Com esta instância foi possível analisar, em cada momento, se o modelo estava a ser eficaz, sem exigir grande capacidade computacional. Após a validação do modelo com instância de teste, submeteu-se o mesmo a instâncias mais representativas da realidade com maior volume de dados. As etapas necessárias para a construção das instâncias são descritas detalhadamente de seguida. Nesta secção é também descrito o formato das folhas Excel criadas de acordo com os requisitos exigidos para a implementação em Python dos algoritmos necessários.

4.2.1 Geração dos Dados de Teste

Numa primeira fase foi necessário obter uma instância de teste com dados representativos do Shopstocker. Esta instância foi gerada de forma a ser grande o suficiente para permitir avaliar o comportamento do modelo, mas razoavelmente pequena para não aumentar a exigência computacional requerida para o resolver. Desta forma, os dados foram gerados num ficheiro Excel, assumindo que:

- O armazém era composto por 13 linhas;

- Em cada linha existiam 10 posições;
- Existiam 15 referências diferentes correspondentes a 15 tipos de PC;
- Todas as 130 posições do armazém estavam preenchidas por um PC;
- A cada PC foi associado um identificador numérico único, de 1 a 130.
- A distribuição das 15 referências pelas 130 posições do armazém foi gerada aleatoriamente, através da função *RANDBETWEEN()* do Excel. Cada referência teve a mesma probabilidade de surgir em cada posição do armazém.
- A sequência de saída tinha 10 posições;
- A ordem das referências na sequência de extração foi gerada aleatoriamente com auxílio da função *RANDBETWEEN()* do Excel. Cada referência teve a mesma probabilidade de surgir em cada posição da sequência.
- Cada para-choques tinha um valor único de FIFO aleatoriamente atribuído, indicando para cada referência, a antiguidade relativa dos PC. O valor de 1 representava o PC mais antigo da referência, sendo que quanto mais elevado o valor de FIFO, mais recente era o PC.

Após a geração destes dados, criaram-se duas folhas num ficheiro Excel, denominado "*dados.xlsx*". A necessidade de utilizar duas folhas distintas prende-se com o facto dos *data frames* importados e utilizados no Python necessitarem de ter o mesmo número de registos para cada identificador. Como os dados relativos às referências, linhas e posição estão associados a um identificador de PC único, entre 1 e 130, podem ser importados a partir da mesma folha Excel. No entanto, como as referências de cada posição da sequência de extração apenas estão definidas para as 10 posições da mesma, foi necessária uma nova folha e a consequente importação de um diferente *data frame*. A informação existente em cada folha criada será descrita de seguida.

Os dados da primeira folha, *armazem*, dos quais oito podem ser observados na Figura 11, foram distribuídos por cinco colunas, *cod*, *refA*, *linA*, *posA* e *fifo*. A coluna *cod* contêm o identificador único de cada um dos 130 PC, enquanto que, as restantes colunas comportam a informação alusiva às referências, linhas de armazenamento, posição de armazenamento e FIFO de cada PC, respetivamente.

	A	B	C	D	E
1	cod	refA	linA	posA	fifo
2	1	1	10	1	5
3	2	1	1	2	6
4	3	1	6	3	3
5	4	1	4	4	4
6	5	1	12	6	1
7	6	1	6	8	2
8	7	1	2	9	7
9	8	1	12	9	8

Figura 11 - Dados referentes à folha armazem do ficheiro Excel "dados.xlsx.

Na folha *refSeq* os dados das referências das dez posições da sequência de extração foram dispostos em quatro colunas, ilustradas na Figura 12. A primeira coluna, identificada como *ord* é um identificador único da ordem da posição da sequência. Já a coluna *refS* inclui a referência requerida em cada posição da sequência. De modo a facilitar o acesso à posição da sequência na construção do modelo na linguagem de programação Python, desenvolveu-se a coluna *id* com dados iguais aos da coluna *ord*. Por fim, a coluna *seqfifo* contempla o valor que é expectável ter em cada posição da sequência para que se cumpra a regra FIFO.

	A	B	C	D
1	ord	refS	id	seqfifo
2	1	3	1	1
3	2	15	2	1
4	3	10	3	1
5	4	10	4	2
6	5	3	5	2
7	6	13	6	1
8	7	13	7	2
9	8	5	8	1
10	9	9	9	1
11	10	12	10	1

Figura 12 - Dados referentes à folha refSeq do ficheiro Excel "dados.xlsx".

Os dados contidos neste ficheiro Excel foram posteriormente importados para o ambiente de desenvolvimento integrado PyCharm. A importação e manipulação dos mesmos é descrita detalhadamente no Anexo I presente na página 99.

4.2.2 Construção da Configuração 1

Neste projeto de dissertação privilegiou-se a configuração ideal de partida, isto é, a situação de início do dia, uma vez que é a única situação que faz sentido para realizar comparações orientadas às sequências de extração. O estado do Shopstocker deve ser mantido o mais próximo possível da situação ideal e por isso deve retomar a sua configuração inicial. Qualquer outra configuração do armazém ajustada a uma particular sequência seria difícil de obter uma vez que existem entradas, saídas e realocações a todo o instante no Shopstocker e a sequência seguinte é impossível de prever.

Os pontos seguintes descrevem as diversas etapas realizadas, desde a fase inicial de análise dos dados fornecidos pela PB, até à construção da configuração do armazém. A Figura 13 representa a organização das posições e das linhas ao longo do armazém.

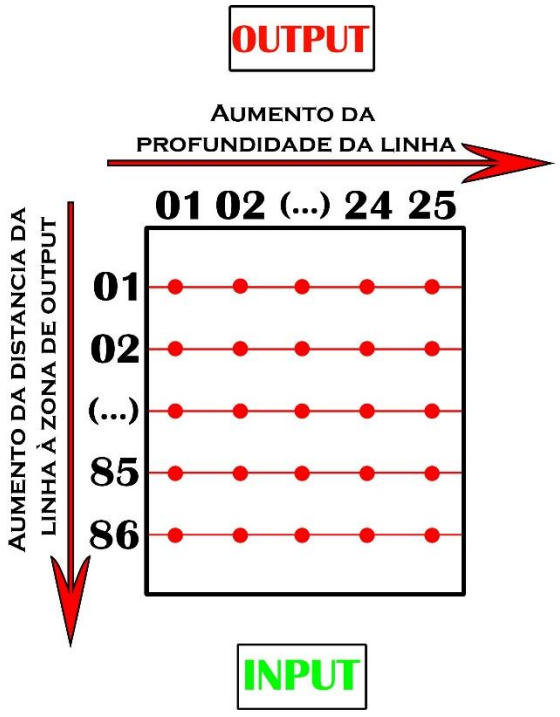

Figura 13 - Identificação das linhas e das posições do Shopstocker.

O índice das posições varia entre os valores 1 e 25, sendo que, o valor 1 corresponde à posição que pode imediatamente ser recolhida, enquanto que qualquer posição $1 < p <= 25$ implica que sejam recolhidas previamente as, $p - 1$, posições. Por sua vez, o valor associado às linhas de 1 a 43 no piso inferior e 44 a 86 no piso superior aumenta à medida que a linha se afasta da zona de *output*. Estes valores relativos às posições e às linhas do Shopstocker foram utilizados apenas como identificadores e não representam distâncias ou tempos de extração relativamente à zona de *output*. No entanto, índices

maiores representarão distâncias e tempos maiores da seguinte forma: retirar um PC da posição *p* da linha 23 demorará mais do que extrair um PC da posição *p* da linha 7. De igual modo, extrair um PC da posição 18 da linha / demorará mais do que extrair um PC da posição 5 da mesma linha, até porque todos os suportes das posições 1... 17, incluindo o 5, terão que ser retirados.

Análise ABC

Após a receção dos ficheiros de dados provenientes da PB, realizou-se uma verificação inicial no sentido de detetar eventuais falhas ou valores nulos. Com a informação devidamente validada optou-se pela seleção dos dados alusivos aos PC frontais. A Tabela 1 ilustra os primeiros registos, e as diversas colunas do ficheiro de dados fornecido, de onde se destacam as seguintes: *Hook*, identificador do suporte; *Unload Timestamp*, data e hora da extração do PC; *Part Number*, número identificador do PC; *Output*, saída onde o PC foi extraído; *Sequency*, posição na sequência de extração do PC; Reference, referência identificadora do modelo do PC e, *Empty Hook*, que informa, através de uma sistema binário, se um suporte chegou vazio à zona de *output*.

Tabela 1 - Dados referentes às saídas de PC frontais, fornecidos pela PB.

FRONTS						
Hook	Unload Timestamp	Part Number	Output	Sequency	Reference	Empty Hook
32224	03/01/2020 07:10	1528M19C8M411401	6	447	7206340100182	0
30251	03/01/2020 07:15	1528M19C6M044901	4	448	7210000100182	0
30379	03/01/2020 07:16	1528M19CB1273901	4	449	7214030100128	0
31922	03/01/2020 07:19	1528M19CB2355401	4	451	7214030100199	0
31765	03/01/2020 07:20	1528M19CBC285901	4	452	7210000100198	0
32382	03/01/2020 07:21	1528M19CAN271301	4	453	7214030100128	0
30109	03/01/2020 07:25	1528M19C6G094301	4	454	7214030100226	0
31201	03/01/2020 07:27	1528M19CB6492401	4	456	7210000100198	0
31885	03/01/2020 07:28	1528M19CAD593101	4	458	7210000100222	0
31209	03/01/2020 07:28	1528M19CB3242801	4	459	7219010100128	0
31845	03/01/2020 07:30	1528M19AP2370801	4	461	7214030100198	0
30858	03/01/2020 07:32	1528M19B60452901	6	450	7236590100207	0
31928	03/01/2020 07:36	1528M19CAF560701	4	462	7210000100222	0
31788	03/01/2020 07:36	1528M19C9C120801	4	464	7214030100223	0
30115	03/01/2020 07:39	1528M19CB6295401	6	455	7219000100199	0
31707	03/01/2020 07:39	1528M19C6D321401	4	465	7214030100198	0

De acordo com a informação recolhida, a empresa realiza mensalmente uma avaliação ABC às saídas de PC de modo a atualizar as referências pertencentes às três classes: *high runners*, *medium runners* e *low runners*. Por esta razão, optou-se por selecionar os dados mais recentes disponíveis compreendidos no período entre os dias 3 de janeiro e 7 de fevereiro de 2020.

Posteriormente, e por simplificação de processos produziu-se uma tabela de conversão de referências representada na Tabela 2, que a cada referência original atribuiu um código a ser usado no modelo desenvolvido.

Tabela 2 - Tabela de conversão das referências.

Referência	Código	Total	Referência	Código	Total	Referência	Código	Total	Referência	Código	Total	Referência	Código	Total
7214030100128	1	1634	7214030100142	23	146	7214030100209	45	59	7219000100142	67	24	7184750100199	89	9
7214030100182	2	1395	7236590100128	24	136	7219000100226	46	59	7236590100182	68	23	7219000100220	90	9
7210000100198	3	1305	7236590100251	25	110	7184750100182	47	58	7236590100199	69	22	13013451	91	9
7214030100198	4	1040	7214030100217	26	106	7206520100128	48	54	7236590100209	70	22	7206520100142	92	8
7210000100111	5	899	7219000100222	27	105	7219010100222	49	53	7214030100203	71	21	7219010100203	93	7
7214030100223	6	676	7206340100222	28	102	7206340100111	50	50	7184750100222	72	17	7219010100226	94	7
7210000100223	7	550	7210000100203	29	99	7206340100199	51	50	7206340100203	73	16	13013327	95	7
7210000100222	8	528	7219000100182	30	99	7219010100111	52	49	7206520100217	74	16	7206340100226	96	6
7210000100182	9	433	7219010100182	31	95	7219000100199	53	42	13013329	75	16	7184750100208	97	6
7219000100198	10	425	7210000100220	32	92	7219010100199	54	42	7206520100111	76	15	7184750100220	98	4
7206340100198	11	363	7206340100217	33	90	7206340100142	55	41	7206520100209	77	15	7206520100226	99	3
7210000100142	12	334	7206340100223	34	89	7219000100208	56	40	13013808	78	15	7206520100203	100	3
7206340100182	13	325	7184750100198	35	88	7184750100111	57	39	7184750100224	79	14	13012990	101	3
7210000100224	14	308	7210000100226	36	84	7219000100111	58	36	7206520100222	80	14	7184750100226	102	1
7210000100199	15	303	7219000100224	37	82	7219010100223	59	33	7219000100203	81	13			
7214030100222	16	292	7219010100217	38	81	7184750100223	60	33	7206520100199	82	13			
7206340100128	17	282	7219000100223	39	77	7236590100217	61	30	13013457	83	13			
7214030100199	18	258	13013463	40	75	7206520100182	62	29	7184750100221	84	12			
7214030100111	19	248	7210000100208	41	68	7219010100209	63	29	7184750100142	85	11			
7219010100198	20	237	7236590100207	42	67	7219000100229	64	28	13014080	86	11			
7236590100198	21	217	7214030100226	43	65	7206340100209	65	26	7206520100223	87	10			
7219010100128	22	188	7206520100198	44	59	7219010100142	66	26	7184750100203	88	9			

A Tabela 2 ilustra as três colunas, dispostas em 5 segmentos para facilitar a visualização, desenvolvidas pela seguinte ordem: *Referência*, que identifica a referência original do modelo do PC; *Código*, que contempla o novo valor atribuído à referência e *Total*, que contabiliza o somatório dos registos de cada referência nos dados originais.

Para a atribuição do código identificaram-se de forma única 102 referências distintas no período analisado. Através da função *COUNTIF()* contabilizaram-se o número total de saídas de cada modelo e procedeu-se à sua ordenação por ordem decrescente como a Tabela 2 demonstra.

Por fim, atribui-se na coluna Código um valor inteiro entre 1 e 102 para cada uma das referências devidamente ordenadas. Com esta metodologia o código atribuído representa a importância relativa de cada modelo, sendo que quanto menor o valor, maior a importância da referência em termos de taxa de consumo.

Após a seleção dos dados do período pretendido e a atribuição do código de correspondência, realizou-se uma análise ABC com o intuito de determinar quais as referências que deveriam pertencer a cada classe. Esta técnica permite categorizar os produtos atendendo à sua importância relativa, através de um critério de definição de limites entre cada classe utilizada. A Tabela 3 ilustra os primeiros registos da análise efetuada. A primeira coluna da referida tabela, identifica de forma única a referência em

questão. A coluna *Total* contempla o somatório das saídas da referência no período analisado. A terceira coluna representa o somatório acumulado das referências. Por sua vez, a coluna *% Saídas* exibe a percentagem de extrações da referência em análise, em relação ao total de saídas de PC no período analisado. A quinta coluna mostra o somatório acumulado das percentagens de saída das referências. A coluna *% Ref Acum.* ostenta os valores da percentagem acumulada das referências em relação ao número total de referências existentes. Por fim, a coluna *% Ref* da Tabela 3 exibe a percentagem de cada referência em relação ao total de referências existentes.

Tabela 3 - Análise ABC aos dados de saídas de PC fornecidos pela PB.

Código	Total	Total Acum.	% Saídas	% Saídas Acum.	% Ref Acum.	% Ref
1	1634	1634	10,48%	10,48%	0,98%	0,98%
2	1395	3029	8,95%	19,44%	1,96%	0,98%
3	1305	4334	8,37%	27,81%	2,94%	0,98%
4	1040	5374	6,67%	34,48%	3,92%	0,98%
5	899	6273	5,77%	40,25%	4,90%	0,98%
6	676	6949	4,34%	44,59%	5,88%	0,98%
7	550	7499	3,53%	48,12%	6,86%	0,98%
8	528	8027	3,39%	51,50%	7,84%	0,98%
9	433	8460	2,78%	54,28%	8,82%	0,98%
10	425	8885	2,73%	57,01%	9,80%	0,98%
11	363	9248	2,33%	59,34%	10,78%	0,98%
12	334	9582	2,14%	61,48%	11,76%	0,98%
13	325	9907	2,09%	63,57%	12,75%	0,98%
14	308	10215	1,98%	65,54%	13,73%	0,98%
15	303	10518	1,94%	67,49%	14,71%	0,98%
16	292	10810	1,87%	69,36%	15,69%	0,98%
17	282	11092	1,81%	71,17%	16,67%	0,98%
18	258	11350	1,66%	72,83%	17,65%	0,98%
19	248	11598	1,59%	74,42%	18,63%	0,98%

Tendo por base a Tabela 3 realizou-se uma análise seguindo os limites definidos pela regra de Pareto, na qual 20% dos produtos são responsáveis por 80% das saídas. Assim, definiram-se 22 referências para a classe dos *high runners*. Posteriormente, estabeleceu-se o valor de 55% para a percentagem acumulada das referências em relação ao número total de referências existentes para a fronteira entre as duas classes restantes. Esta percentagem correspondeu a 95% das saídas totais de PC e resultou em 35 referências *medium runners* e 45 da classe *low runners*.

Análise das Referências por Linha

Um outro ficheiro de dados analisado contém informação relativa ao posicionamento das diversas referências *high* e *medium runners* ao longo Shopstocker. De forma similar ao anteriormente descrito, selecionou-se a informação alusiva aos PC frontais. A Tabela 4 apresenta os dados fornecidos, onde a primeira coluna, *Lanes*, indica o identificador da empresa para cada linha de armazenamento. Na segunda coluna da mesma tabela, são identificadas as referências que se podem armazenar na linha indicada. A Tabela 4 exibe os primeiros 13 registos fornecidos para os PC frontais.

Tabela 4 - Dados alusivos às referências por linha, fornecidos pela PB.

FRONTS	
Lanes	References
3	EMPTY HOOKS
4	EMPTY HOOKS
5	EMPTY HOOKS
6	EMPTY HOOKS
7	7210000100198
7	7214030100198
7	7210000100199
7	7214030100199
7	7206340100128
7	7210000100222

Para facilitar o estudo às linhas do Shopstocker procedeu-se à conversão das referências de acordo com a Tabela 2. Para uma organização mais simplificada das linhas de armazenamento, converteu-se a sua identificação original para o código exposto na Tabela 5. O valor de correspondência das linhas foi definido com base nas posições relativas ilustradas na Figura 13. Após as conversões supracitadas, estudou-se a organização das referências ao longo do Shopstocker.

A investigação efetuada permitiu perceber que, todas as referências da classe *high runner* possuíam uma linha dedicada exclusiva para cada modelo no primeiro piso, na zona mais próxima da saída. O estudo revelou ainda que, 8 referências desta classe usufruíam de uma segunda linha dedicada no mesmo piso. Das restantes linhas do primeiro piso, 9 eram caóticas, destinadas a armazenar *high runners* que, quando entraram no sistema não tinham espaço disponível nas linhas dedicadas e exclusivas ao modelo. Por fim, na região do primeiro piso mais próxima da zona de *input*, estavam reservadas 4 linhas destinadas a armazenar suportes vazios.

Da mesma forma, o estudo realizado permitiu inferir que 9 linhas do segundo piso estavam dedicadas a armazenar PC de referências da classe *medium runners*. Nestas linhas podem ser misturadas até três modelos pertencentes a esta classe. Estas linhas localizavam-se na zona mais próxima do elevador de subida, isto é, na região mais afastada da extração do segundo piso.

Das restantes linhas existentes no segundo piso, duas, de capacidade mais reduzida, estavam destinadas ao pré-sequenciamento de PC. As restantes linhas estavam reservadas a suportes vazios. Com esta análise foi possível depreender a distribuição das referências das classes *high* e *medium runners* ao longo do Shopstocker.

Tabela 5 - Tabela de correspondência das linhas originais com o código definido.

Lanes	Código	Lanes	Código	Lanes	Código	Lanes	Código
45	1	23	23	151	45	129	67
44	2	22	24	150	46	128	68
43	3	21	25	149	47	127	69
42	4	20	26	148	48	126	70
41	5	19	27	147	49	125	71
40	6	18	28	146	50	124	72
39	7	17	29	145	51	123	73
38	8	16	30	144	52	122	74
37	9	15	31	143	53	121	75
36	10	14	32	142	54	120	76
35	11	13	33	141	55	119	77
34	12	12	34	140	56	118	78
33	13	11	35	139	57	117	79
32	14	10	36	138	58	116	80
31	15	9	37	137	59	115	81
30	16	8	38	136	60	114	82
29	17	7	39	135	61	113	83
28	18	6	40	134	62	112	84
27	19	5	41	133	63	111	85
26	20	4	42	132	64	110	86
25	21	3	43	131	65		
24	22	152	44	130	66		

Estudo do *Stock* Existente

Na última etapa da análise dos dados estudou-se o *stock* disponível de cada referência no dia 7 de fevereiro de 2020. A Tabela 6 ilustra os 10 primeiros registos. Na primeira coluna é possível observar a referência original e, na segunda, a quantidade de PC em *stock*, em unidades. Para agilização do estudo, converteram-se as referências da primeira coluna através do código correspondente exposto na Tabela 2. O estudo destes dados forneceu uma perceção da distribuição das unidades de PC pelas diferentes referências e classes.

Tabela 6 - Dados sobre o stock de cada referência.

Em stock	
7214030100128	149
7214030100198	101
7236590100182	96
7206340100198	89
7210000100223	72
7219000100198	58
7214030100223	55
7210000100198	47
7214030100182	40
7214030100199	36

Desenvolvimento da Configuração 1

Após a conclusão das três etapas descritas nos pontos acima percebeu-se o número de referências de cada classe, a sua distribuição ao longo do Shopstocker e a quantidade de PC por referência. Com a informação recolhida foi possível construir uma configuração, que se aproximasse da situação previsível do estado do Shopstocker. A configuração 1 está representada na Figura 14.

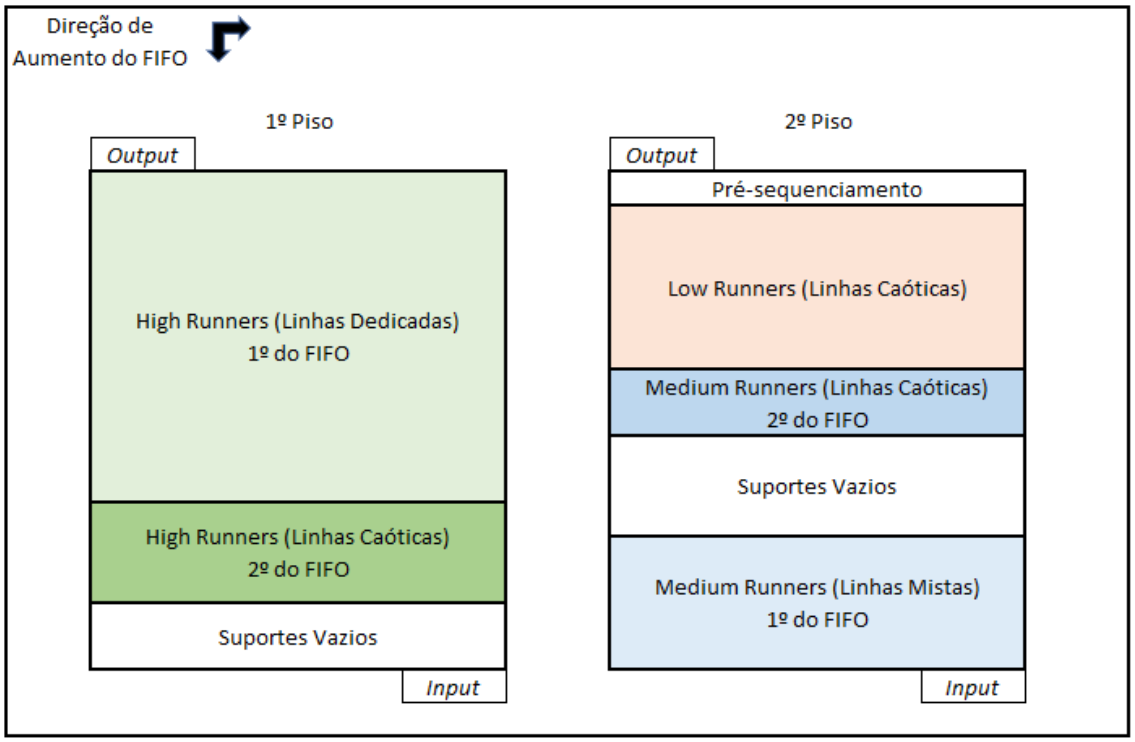

Figura 14 - Organização das classes pelas diferentes zonas do Shopstocker na configuração 1.

No piso inferior foram destinadas 22 linhas para cada referência *high runner*. A estas somaram-se mais 8 linhas para cada uma das 8 primeiras referências desta classe. Por fim, geraram-se de forma aleatória, com auxílio da função *RANDBETWEEN()* do Excel, 9 linhas caóticas reservadas a *high runners*. Todas as 22 referências desta classe tiveram a mesma probabilidade de serem alocadas a cada uma das 25 posições destas 9 linhas. As restantes 4 linhas ficaram, tal como mostrado nos dados analisados, destinadas a receber suportes vazios.

Já para a classe dos *medium runners* produziram-se 9 linhas mistas, na região mais afastada da zona de extração do segundo piso, destinadas a receber 25 referências desta classe, e 5 linhas caóticas destinadas a armazenar de forma aleatória as 35 referências desta classe. O número de linhas caóticas foi definido de modo a aproximar os dados gerados ao stock real analisado. Através da função *RANDBETWEEN()* do Excel, devidamente adaptada para cada situação, os 35 modelos desta classe foram alocados às linhas destinadas aos mesmos.

Por fim, reservaram-se ainda 11 linhas caóticas, destinadas ao armazenamento de todos os PC das 45 referências da classe *low runner*. O número de linhas foi definido de modo a aproximar os dados gerados ao stock real analisado e, por outro lado, de forma a garantir que todas as referências estivessem representadas no armazém. Uma vez mais, a função *RANDBETWEEN()* do Excel permitiu gerar de forma aleatória os PC de cada posição.

Todas as posições das restantes linhas foram alocadas com suportes vazios que, por necessidade da linguagem de programação, se convencionou categorizar como referência 103.

Configuração da Informação FIFO

Com os PC distribuídos pelas diversas posições do armazém atribuiu-se um valor de FIFO, seguindo as seguintes regras:

- Todos os PC possuíam um valor de FIFO único dentro de cada referência.
- Quanto menor o valor de FIFO, mais antigo é o PC dentro da sua referência.
- Para cada referência, os valores de FIFO variaram entre 1 e o valor correspondente ao total de PC do respetivo modelo.
- O valor do FIFO das classes *high* e *low runners* foi atribuído aos PC por ordem crescente do seu identificador.
- Os PC mais antigos da classe *medium runners*, primeiros na ordem de FIFO, foram alocados nas linhas mistas dedicadas aos mesmos. Aos restantes PC desta classe, armazenados nas linhas caóticas, foram atribuídos valores de FIFO maiores.

A distribuição dos valores de FIFO atribuídos aos PC ao longo do Shopstocker pode ser observada na Figura 14. Tal como a citada figura ilustra, os *high runners* mais antigos, primeiros da ordem FIFO, encontram-se na região mais próxima da zona de *output* do primeiro piso. Por sua vez, os *high runners* mais recentes, segundos da ordem FIFO, foram alocados à região do primeiro piso mais afastada da zona de saída. Já os PC da classe *medium runner* mais antigos, primeiros da ordem FIFO, foram colocados nas linhas mistas existentes no segundo piso. Os PC mais recentes desta classe, e por isso segundos na ordem FIFO, foram alocados às linhas caóticas do piso superior.

Construção do Ficheiro Excel "*conf_01.xlsx*"

Para a importação dos dados gerados para o ambiente de desenvolvimento integrado PyCharm, foi necessário definir os mesmos no formato adequado, no ficheiro Excel "*conf_01.xlsx*". Neste documento Excel foram criadas 3 folhas, *armazem*, *temposLin* e *temposPos*, cujo conteúdo será descrito detalhadamente de seguida. A Figura 15 apresenta os 15 primeiros registos guardados na folha *armazem* do ficheiro Excel citado.

	A	B	C	D	E	F	G	H
1	cod	refA	linA	posA	fifo	textracao	tLin	tPos
2	1	1	1	1	1	8	0	8
3	2	1	1	2	2	16	0	16
4	3	1	1	3	3	24	0	24
5	4	1	1	4	4	32	0	32
6	5	1	1	5	5	40	0	40
7	6	1	1	6	6	48	0	48
8	7	1	1	7	7	56	0	56
9	8	1	1	8	8	64	0	64
10	9	1	1	9	9	72	0	72
11	10	1	1	10	10	80	0	80
12	11	1	1	11	11	88	0	88
13	12	1	1	12	12	96	0	96
14	13	1	1	13	13	104	0	104
15	14	1	1	14	14	112	0	112
16	15	1	1	15	15	120	0	120

Figura 15 - Folha, armazem, do ficheiro Excel "conf_01.xlsx".

A primeira coluna da folha em questão, *cod*, contempla o identificador único do PC que variou entre 1 e 2150. A segunda coluna aporta a referência correspondente ao PC indicado. Por sua vez, as colunas

linA e *posA* contêm, respetivamente, os identificadores da linha e da posição do PC identificado. Os valores de FIFO associados a cada PC foram inseridos na coluna *fifo*. Por fim, os dados relativos ao tempo de extração de cada PC, foram distribuídos por três colunas: *textracao*, que contempla o somatório do tempo de extração da linha e da posição do PC, *tLin*, que apresenta o tempo associado à linha do PC em questão e *tPos*, que ostenta os valores associados ao tempo de extração do PC indicado da sua posição no Shopstocker.

Adicionalmente, geraram-se duas folhas com os tempos de extração associados às 86 linhas e a cada uma das 25 posições das mesmas.

A Figura 16 apresenta os oito primeiros registos da folha *temposLin* distribuídos por duas colunas, *idLin* e *lin*. Na primeira coluna foi identificada de forma única cada uma das 86 linhas. Já na segunda coluna, *lin*, estão associados os tempos de extração, em segundos, da linha indicada.

	A	B
1	idLin	lin
2	1	0
3	2	5
4	3	10
5	4	15
6	5	20
7	6	25
8	7	30
9	8	35

Figura 16 - Folha, temposLin, do ficheiro Excel "conf_01.xlsx".

De forma análoga, a Figura 17 apresenta em duas colunas os tempos de extração alusivos às primeiras oito posições das linhas de armazenamento. A primeira coluna, *idPos*, identifica de forma única cada uma das 25 posições. Na segunda coluna, *pos*, foi guardada a informação, em segundos, do tempo de extração associado à posição identificada.

	A	B
1	idPos	pos
2	1	8
3	2	18
4	3	28
5	4	38
6	5	48
7	6	58
8	7	68
9	8	78

Figura 17 - Folha, temposPos, do ficheiro Excel "conf_01.xlsx".

4.2.3 Construção da Configuração 2

A segunda configuração definida neste projeto teve como base o estudo feito para definir a configuração 1. Numa tentativa de diminuir a mistura de referências nas linhas caóticas e mistas, procurou-se, nesta segunda configuração, distribuir as referências *low* e *medium runners* por um maior número de linhas. A Figura 18 apresenta a alocação definida dos PC às diferentes posições assim como dos valores de FIFO atribuídos.

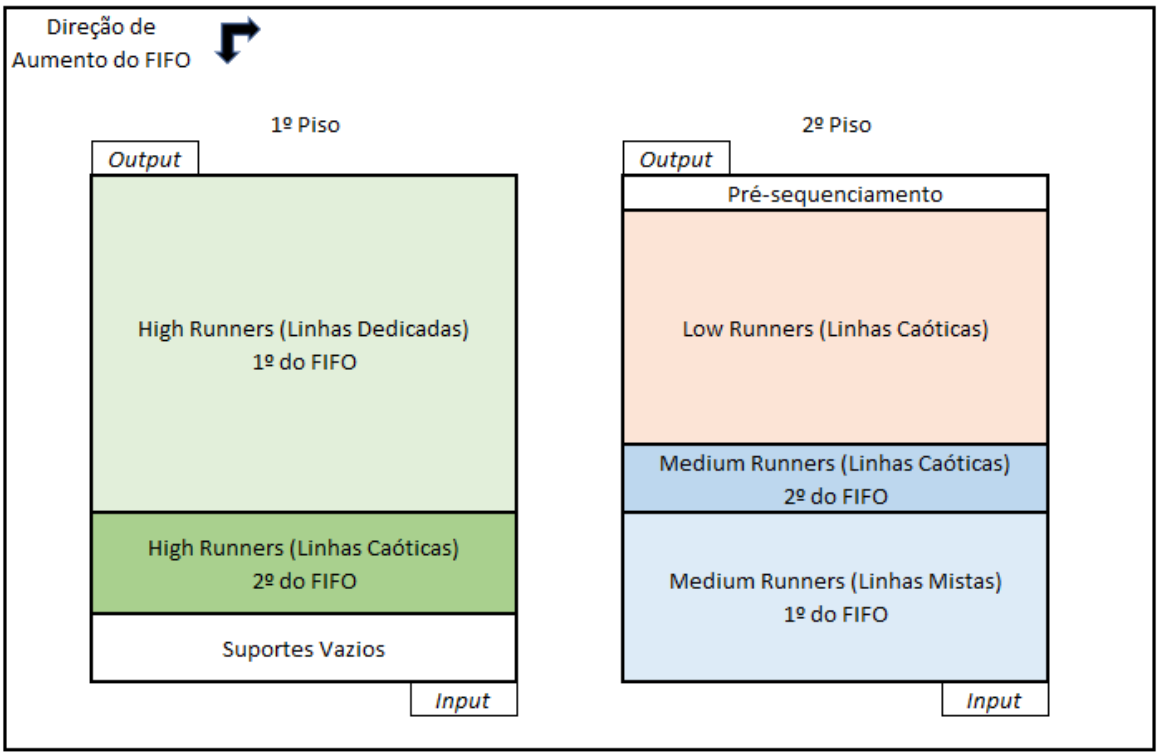

Figura 18 - Organização das classes pelas diferentes zonas do Shopstocker na configuração 2.

Tal como a Figura 18 ilustra, o primeiro piso é uma réplica integral da configuração 1. Os *high runners* foram igualmente distribuídos pelas linhas do piso inferior, mantendo a ordem de FIFO anteriormente definida. Por outro lado, numa tentativa de retirar proveito das linhas alocadas inteiramente com suportes vazios, dividiram-se estes vazios pelas linhas caóticas e mistas destinadas a *low* e *medium runners*. Com esta nova configuração, diminui-se a mistura de referências nas linhas contendo estes modelos. A quantidade de PC em stock de cada referência foi similar à gerada na configuração 1.

Para importar os dados para o ambiente de desenvolvimento integrado PyCharm, definiu-se a mesma estrutura do documento Excel "*conf_01.xlsx*" descrito anteriormente. Assim, os novos dados relativos às folhas *armazem*, *temposLin* e *temposPos*, foram guardados e carregados para o PyCharm a partir do documento "*conf_02.xlsx*".

4.2.4 Construção das Sequências de Extração

Para avaliar as configurações desenvolvidas geraram-se sequências de extração a partir dos dados fornecidos. Uma vez que os dados foram registados de forma contínua ao longo do tempo, não foi possível depreender dos mesmos onde se iniciou ou terminou cada sequência de carregamento dos camiões. Por esta razão, e após a validação por parte da empresa, foi definido um valor constante de 100 unidades para cada sequência de extração. Com este valor definido, gerou-se uma macro em VBA, ilustrada na Figura 19, que gerou 155 sequências de 100 PC cada, a partir dos registos iniciais.

```
Sub copiar()

Dim linha As Integer
Dim coluna As Integer
Dim i As Integer

linha = 2
coluna = 7

Do Until Cells(1, coluna).Value = Empty
    For i = 2 To 101
        Cells(linha, 1).Select
        Selection.Copy
        Cells(i, coluna).Select
        Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, SkipBlanks _
            :=False, Transpose:=False
        linha = linha + 1
    Next i
    coluna = coluna + 1
Loop

End Sub
```

Figura 19 - Código da macro desenvolvida em VBA.

De forma a poder analisar as sequências de extração geradas em tempo útil, optou-se por selecionar aleatoriamente uma amostra de 30 sequências a partir das 155 geradas. Cada sequência selecionada foi posteriormente guardada no ficheiro Excel "*sequencias.xlsx*" na folha correspondente. A Figura 20 mostra os 8 primeiros registos guardados, relativos à sequência de extração 1. A primeira coluna, *ord*, identifica de forma única a posição da sequência de extração. A coluna *refS* ostenta a referência pretendida na posição indicada. Para facilitar a manipulação dos dados na linguagem de programação, foi adicionada a coluna *id* que possui os mesmos valores da coluna *ord*. Por fim, com o intuito de analisar o cumprimento do FIFO, foi gerada a coluna *seqfifo* contendo os valores de FIFO esperados para que, em cada posição da sequência, o FIFO fosse cumprido.

	A	B	C	D
1	ord	refS	id	seqfifo
2	1	13	1	1
3	2	9	2	1
4	3	1	3	1
5	4	18	4	1
6	5	3	5	1
7	6	1	6	2
8	7	43	7	1
9	8	3	8	2

Figura 20 - Folha, seq1, do ficheiro Excel "sequencias.xlsx".

Para cada sequência de extração foi criada uma folha com uma estrutura igual à da Figura 20, cujo nome identificava de forma única a sequência em questão.

Adicionalmente, desenvolveram-se sequências de teste e outras adequadas a determinadas experiências computacionais que foram igualmente guardadas no mesmo documento Excel, "*sequencias.xlxs*".

4.3 Análise Combinatória do Problema Real

A análise combinatória permite perceber o número de possibilidades (combinações) para a ocorrência de um determinado evento. Pelo princípio fundamental da contagem é possível determinar o número total de combinações através de fórmulas específicas dependendo da situação em estudo. A análise combinatória define diversas situações das quais se destacam as seguintes:

- Permutações simples – número de combinações pelas quais podemos ordenar n elementos diferentes.
- Combinações simples – número de combinações de k elementos de um conjunto total de n elementos sem repetição, sem ter em consideração a ordem desses elementos.
- Arranjos simples – número de combinações de k elementos de um conjunto de n elementos sem repetição, atendendo à ordem desses elementos.
- Arranjos com repetição – número de combinações de k elementos de um conjunto de n elementos com repetição, atendendo à ordem desses elementos.

Com o número total de combinações possíveis para cada evento, é possível perceber a complexidade computacional que um determinado problema exige para o resolver. Um algoritmo de enumeração completa calcula todas as possibilidades para um determinado evento e determina a melhor. Em problemas simples e com um reduzido volume de dados a solução ótima é fácil de obter com este método. No entanto, os problemas reais originam diversas combinações possíveis e, consequentemente, exigem maior capacidade computacional para os resolver. Nesta secção procurou-se comparar a complexidade do problema real estudado existente no Shopstocker com problemas amplamente estudados e descritos na literatura.

4.3.1 Euromilhões

No jogo do Euromilhões o jogador tem de escolher uma chave composta por 5 números (k1) de um total de 50 bolas (n1) e 2 estrelas (k2) de um total de 12 existentes (n2). Neste jogo a ordem da sequência de números e estrelas não é relevante e não é permitida a sua repetição. Tratando-se de uma situação de combinações simples e pela aplicação da fórmula definida, o número total de combinações existentes para este jogo é:

$$C_{n1,k1} \times C_{n2,k2} = C_{50,5} \times C_{12,2} = \frac{50!}{5!\ (50-5)!} \times \frac{12!}{2!\ (12-2)!} = 139838160$$

Uma vez que o jogador necessita de acertar nos 5 números e 2 estrelas corretos antes da sua extração o jogo é extremamente difícil devido ao elevado número de combinações possíveis. Por outro lado, como cada número e cada estrela são exclusivos nos conjuntos de n1 e n2 elementos, se a chave for previamente conhecida apenas uma combinação é possível e a solução para este problema é imediata.

4.3.2 MasterMind

O MasterMind é um jogo cujo objetivo é descobrir uma sequência de cores definida pelo oponente através de palpites sucessivos. Após um palpite do jogador, o oponente utiliza um marcador preto para identificar uma cor da sequência na posição correta. No caso de a cor estar na posição errada o oponente assinala-a com um marcador branco. Caso a cor selecionada pelo jogador não faça parte da sequência nenhum marcador é utilizado pelo oponente. Após analisar o código de resposta do oponente, o jogador faz um novo palpite e obtém uma nova resposta. O jogo termina caso o jogador decifre a sequência correta de cores ou se um número limite de palpites for atingido. Na sua versão mais simples o MasterMind utiliza um conjunto de 6 cores diferentes (n) e uma combinação de 4 dessas cores (k).

Se não for permitida a repetição de cores e aplicando a fórmula dos arranjos simples existem as seguintes combinações:

$$A_{n,k} = A_{6,4} = \frac{6!}{(6-4)!} = 360$$

Por outro lado, se a repetição de cores for permitida e aplicando a fórmula dos arranjos com repetição existem as seguintes combinações:

$$A_{n,k} = A_{6,4} = 6^4 = 1296$$

Estes resultados demonstram que se existir mais do que uma possibilidade para cada cor, o número de combinações aumenta consideravelmente em relação à situação de um elemento exclusivo por cor. De forma análoga, a tarefa de acertar na sequência correta torna-se bastante complexa, à medida que mais cores são utilizadas no jogo. Assim, verifica-se que o aumento de cores e a sua repetição no conjunto inicial, tornam este problema difícil de resolver devido ao elevado número de combinações possíveis.

4.3.3 TSP

No problema do Caixeiro Viajante, em inglês *Travelling Salesman Problem* (TSP), um vendedor inicialmente na sua cidade, tem de visitar um conjunto de cidades de forma única e regressar à cidade inicial. O vendedor conhece a distância entre todas as cidades e procura determinar qual a melhor sequência para minimizar a distância total percorrida (de Koster et al., 2007). Este problema enquadra-se na situação das permutações simples definida. Assim, pela aplicação da fórmula existem as seguintes possibilidades para n cidades:

$$P_n = n!$$

À medida que o número de elementos cresce o número de combinações aumenta de forma exponencial o que inviabiliza a sua resolução em tempo útil. O TSP é um problema NP-difícil e não foi encontrado nenhum algoritmo capaz de o resolver em tempo polinomial.

4.3.4 Shopstocker

Considere-se o problema real estudado no Shopstocker como a escolha de uma sequência de 10 PC devidamente ordenados, a partir de um conjunto de 200 armazenados, igualmente distribuídos por 4 referências (A, B, C e D). Considerando-se que as 4 referências se encontram aleatoriamente distribuídas pelas posições do armazém, é necessário selecionar um PC de cada referência para cada posição da sequência de extração que a requer. Tal como no jogo do MasterMind, a necessidade de garantir a sequência correta e a existência de 50 PC para cada referência aumentam consideravelmente o número de combinações possíveis. Considerando a sequência de referências ABCDABCDAB e pela aplicação da fórmula dos arranjos simples temos as seguintes combinações para este problema:

$$A_{50,1} \times A_{50,1} \times A_{50,1} \times A_{50,1} \times A_{49,1} \times A_{49,1} \times A_{49,1} \times A_{49,1} \times A_{48,1} \times A_{48,1} = 8{,}3E + 16$$

Este elevado número de possibilidades para uma instância reduzida de 4 referências e 200 PC revela a complexidade do problema real estudado no Shopstocker.

Por outro lado, a regra FIFO desempenha um papel fundamental neste sistema. Caso seja possível extrair a sequência FIFO, a solução é única e imediata. No entanto, a extração da sequência FIFO pode não ser possível, devido à criação de "*deadlocks*". Se a disposição de armazenamento dos PC da ordem FIFO for inversa à ordem requerida na sequência, a extração de um PC elimina o outro para a solução corrente. Assim, o número de combinações está também dependente da disposição dos PC no armazém e da sequência requerida.

4.3.5 Análise Comparativa

De acordo com a análise individual realizada aos diferentes problemas percebeu-se que a complexidade aumenta consideravelmente com a repetição de elementos e com a relevância da ordem dos mesmos. De igual modo, e tratando-se de problemas com números fatoriais, à medida que o número de elementos cresce, a complexidade do problema aumenta igualmente de forma fatorial. Por esta razão,

um algoritmo de enumeração completa não conseguiria encontrar a melhor solução em tempo útil para instâncias representativas da realidade.

A Figura 21 ilustra o gráfico construído com as combinações possíveis para os problemas estudados.

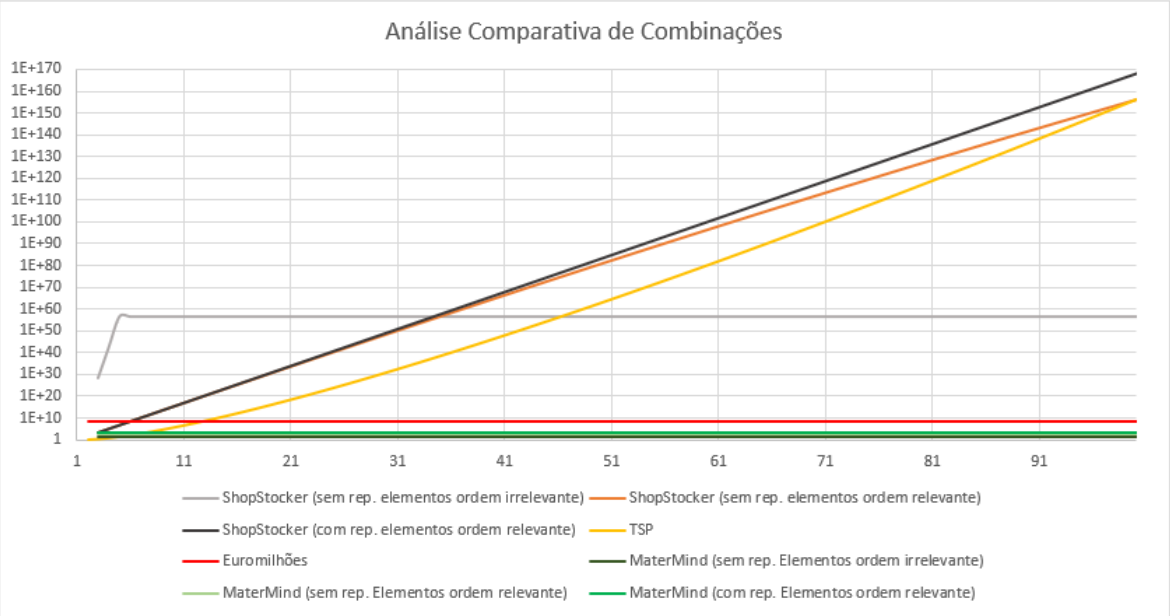

Figura 21 - Análise comparativa do número de combinações possíveis.

De acordo com o gráfico, o número de combinações possíveis para o jogo MasterMind, na sua versão mais simplificada, é reduzido em comparação com os restantes. No jogo do Euromilhões o número de combinações não excede os 1E+10 possibilidades, o que apesar de elevado, é consideravelmente inferior aos restantes problemas. Já no problema TSP verifica-se um aumento exponencial do número de possibilidades à medida que o número de elementos aumenta. Por sua vez, no problema real estudado no Shopstocker verifica-se que, para o caso definido acima e não dando relevância à ordem das referências na sequência, o número de combinações não excederia as 1E+60 possibilidades. No entanto, como a ordem da sequência é um fator decisivo para o correto funcionamento do Shopstocker, o número de combinações possíveis cresce consideravelmente à medida que o número de posições na sequência extração aumenta. Para um n igual a 100 verifica-se que o número de possibilidades do problema real estudado no Shopstocker iguala o número de combinações do problema TSP, atingindo ambos as 1E+160 possibilidades. Atendendo a esta análise, e sabendo que num sistema real existem mais de 2000 PC distribuídos por mais de 100 referências, o número de combinações

possíveis uma sequência de 100 posições é muito elevado. Devido a esta complexidade, um algoritmo de enumeração completa não conseguiria resolver o problema real em estudo em tempo útil.

4.4 Efeito de Compensação nos Tempos de *Setup*

Um dos problemas estudos e apresentados na literatura é o problema de escalonamento SDST *flowshop*, em que a ordem pela qual N trabalhos devem ser processados por M máquinas está sujeita a tempos de *setup* para cada máquina dependentes da sequência (Gupta, 1986). Antes da realização do próximo trabalho n pela máquina m, é necessário preparar a máquina de acordo com os requisitos desse trabalho n atendendo à configuração da máquina m no trabalho corrente. Assim, a melhor sequência de trabalhos deve ter igualmente em consideração os diferentes tempos de *setup* das máquinas, de forma a reduzir o tempo total de processamento dos N trabalhos (Ruiz, Maroto, & Alcaraz, 2005).

No problema real em estudo verifica-se, em determinadas circunstâncias, a necessidade de remover todos os PC à frente (mais próximos da saída) do PC selecionado, que pode ser entendido como tempo de preparação. A título de exemplo, a extração de um PC da posição 20 da linha A obriga a que todos os PC dessa linha nas 19 posições anteriores sejam previamente removidos. Esta tarefa aumenta a entropia do sistema e, consequentemente, o tempo total de extração dos PC pretendidos. Em determinadas situações podem ser selecionados vários PC da mesma linha para a sequência requerida. Nesta situação, a extração do PC mais próximo da extremidade de saída aproxima os restantes das primeiras posições. Por outro lado, se fosse selecionado o PC da posição 10 da linha A, a sua extração, e a dos 9 à sua frente, aproximaria o PC da posição 20 da extremidade de saída. Assim, no instante em que o PC da posição 20 fosse solicitado, a sua posição real seria a 10 e apenas teria de movimentar os 9 à sua frente. A esta particularidade convencionou-se chamar de efeito de compensação. A extração de um PC compensa, em tempo e em número de PC movimentados, a extração de um segundo PC da mesma linha. Assim, e de forma análoga ao problema SDST *flowshop*, no problema real estudado existe frequentemente a necessidade de realizar uma operação prévia antes da extração do PC selecionado. O tempo de *setup* associado à remoção de todos os PC que se encontram à frente do selecionado depende não só da posição de armazenamento, mas também do número de PC selecionados na mesma linha. Esta evidência vai de encontro ao sugerido por Gupta (1986), que afirmou que o tempo de *setup* está dependente igualmente da sequência de trabalhos, uma vez que a preparação da máquina para o trabalho seguinte depende do trabalho corrente. O efeito de compensação existente no Shopstocker tem

igualmente implicações no tempo de *setup*, uma vez que a extração de um PC escolhido diminui o tempo de *setup* necessário para remover outro PC selecionado da mesma linha em posição superior. O tempo de *setup*, e consequentemente o tempo total de extração, está por isso dependente da posição de armazenamento dos PC selecionados e do número de PC que em cada linha fazem parte do carregamento.

4.5 Modelação Matemática do Problema

Considerando o Shopstocker implementado e descrito na secção 4.2.2, assumiram-se as seguintes situações:

- O sistema está em funcionamento sem quebras ou problemas associados à manutenção.
- Todas as 102 referências analisadas estão presentes no armazém.
- Todas as posições de armazenamento estão ocupadas com PC ou suportes vazios.
- O número de PC existente em *stock* é suficiente para suprir uma eventual quebra de fornecimento durante 3 dias.
- Não existe quebra de *stock*.
- O funcionamento do sistema é realizado a uma taxa constante de velocidade.
- Não existem deslizes ou falhas no sistema de tração nem nas escovas.

No armazém em estudo existem N_PC para-choques i dispostos em posições de origem, dos quais é necessário extrair um subconjunto ordenado S que necessitam de ser alocados às j posições destino da sequência de saída, com $j \in S$. O objetivo é preencher todas as j posições de S minimizando o custo da movimentação dos PC i. A formulação matemática deste problema assenta na variável binária X_{ij} e pode ser observada de seguida.

Primeiramente define-se a notação utilizada para este problema.

N_PC – É um parâmetro que representa o número total de PC armazenados.

N_seq – É um parâmetro que representa o número total de posições na sequência de extração.

N_lin – É um parâmetro que representa o número total de linhas de armazenamento existentes no armazém.

N_pos – É um parâmetro que representa o número total de posições em cada linha de armazenamento.

C_{ij} – É um parâmetro que representa o custo de alocação do PC i à posição j.

RA_i – É um parâmetro que representa a referência do PC i armazenado.

RS_j – É um parâmetro que representa a referência do PC pretendido na posição j de S.

SEQ_j – É um parâmetro que representa a posição j da sequência de extração.

POS_i – É um parâmetro que representa a posição de armazenamento do PC i.

$valorFIFO_i$ – É um parâmetro que representa o valor de FIFO do PC i.

$seqFIFO_j$ – É um parâmetro que representa o valor de FIFO esperado na posição j da sequência de extração.

$tempExtrPOS_i$ – É um parâmetro que representa o tempo de extração da posição de armazenamento do PC i.

$tempExtrLIN_l$ – É um parâmetro que representa o tempo de extração da linha de armazenamento l.

N – Representa o conjunto de PC que podem ser selecionados para responder a uma encomenda. Este *set* foi indexado com i, de tal forma que i = 1, 2, ..., N_PC.

S – Representa o conjunto de posições na sequência de extração que necessitam de ser ocupadas. Este *set* foi indexado com j, de tal forma que j = 1, 2, ..., N_seq.

L – Representa o conjunto de linhas de armazenamento existentes no armazém. Este *set* foi indexado com l, de tal forma que l = 1, 2, ..., N_lin.

P – Representa o conjunto de posições existente em cada linha de armazenamento. Este *set* foi indexado com p, de tal forma que p = 1, 2, ..., N_pos.

X_{ij} – Representa a variável de decisão binária, que toma o valor de 1 se o PC i for alocado à posição j de S e 0 no caso contrário.

Z_{lp} – Representa uma variável positiva do tipo inteiro com os índices l, linha de i e p, posição de i, que assume o valor j, caso o PC i seja alocado à posição j de S.

$FIFO_{ij}$ – Representa uma variável positiva do tipo inteiro que assume o valor de FIFO associado ao PC i alocado à posição j de S.

$JFIFO_j$ – É uma variável binária que assume obrigatoriamente o valor de 0, caso o valor da variável $FIFO_{ij}$ seja diferente do esperado na posição j.

$percFIFO$ – É uma variável contínua que toma o valor da percentagem de FIFO cumprida.

Y_{lp} – É uma variável positiva do tipo inteiro com os índices l, linha de i, e p, posição de i.

MOV_l – É uma variável inteira que recebe a posição mais elevada movimentada na linha l.

$totMOV$ – É uma variável inteira positiva que recebe o número total de PC movimentados.

W_{lp} – É uma variável positiva do tipo inteiro com os índices l, da linha de i, e p, da posição de i.

$tempPOS_l$ – É uma variável inteira que recebe o tempo de extração mais elevado na linha l.

$auxTempLIN_{lp}$ – É uma variável binária que assume o valor de 1 caso o PC da linha l e da posição p seja selecionado.

$tempLIN_l$ – É uma variável positiva contínua que recebe o somatório dos tempos de linha de todos os PC selecionados da linha l.

$totLIN_l$ – É uma variável contínua que recebe o valor do tempo de extração total de todos os PC selecionados da linha l.

$totTEMPO$ – É uma variável contínua que contabiliza o tempo total de extração de todos os PC selecionados.

A função objetivo procura minimizar o custo total da alocação dos PC i nas posições j da sequência.

$$Min \sum_{i=1}^{N} \sum_{j=1}^{S} C_{ij} X_{ij} \qquad (1)$$

A primeira restrição desenvolvida garante que cada posição j da sequência é preenchida uma, e uma só vez, por um PC i.

$$\sum_{j=1}^{S} X_{ij} = 1 \ , \qquad \forall\, i \in \mathrm{N} \qquad (2)$$

A segunda restrição deste modelo assegura que cada PC i pode ser atribuído no máximo uma vez, para o conjunto de todas as posições j de S na sequência de extração.

$$\sum_{i=1}^{N} X_{ij} \leq 1 \ , \qquad \forall\, j \in \mathrm{S} \qquad (3)$$

Uma vez que, cada posição j da sequência de saída exige uma referência específica, existe a necessidade de garantir que a referência do PC i alocado é a correta. Desta forma, a terceira restrição assegura que um PC i só é alocado, se a sua referência corresponder à requerida na posição j.

$$RA_i X_{ij} = RS_j X_{ij} \,, \qquad \forall\, i \in \mathrm{N} \qquad \forall\, j \in \mathrm{S} \qquad (4)$$

Do mesmo modo, sendo este armazém constituído por linhas com um fluxo contínuo da extremidade de entrada até à extremidade de saída, é necessário garantir que, se um determinado PC for selecionado para uma posição da sequência, os PC existentes em posições anteriores à posição de armazenamento do selecionado (isto é, mais próximos da saída), não poderão ser alocados a posições superiores na sequência de extração. Esta restrição é necessária porque sempre que um PC é retirado por ser um impedimento físico à extração de outro de uma posição superior da mesma linha, é necessário armazena-lo numa nova posição, que embora sendo desconhecida neste instante, será certamente uma posição de índice 25 e estará em trânsito durante bastante tempo, inviabilizando a sua consideração em termos úteis para a atual sequência em preparação. Por este motivo, a sua nova posição será sempre mais afastada das posições de saída da linha e o modelo matemático elimina este PC das possíveis soluções. Para este efeito desenvolveram-se duas restrições apresentadas de seguida.

A restrição (5) garante que sempre que um PC i é selecionado para uma posição j, ficando a variável X_{ij} com o valor de 1, a variável Z_{lp}, com os índices l, linha de i, e p, posição de i, recebe o valor da posição j da sequência de extração.

$$X_{ij} = 1 \Longrightarrow Z_{(li)(pi)} = SEQ_j \,, \quad \forall\, i \in \mathrm{N} \qquad \forall\, j \in \mathrm{S} \qquad (5)$$

A restrição (6) por sua vez, assegura que o valor da variável Z_{lp} é sempre menor ou igual ao valor da variável Z na posição seguinte da mesma linha, $Z_{l(p+1)}$. Desta forma, um PC A extraído por ser um impedimento físico à extração de B não poderá ser alocado a uma posição j superior à de B.

$$Z_{lp} \leq Z_{l(p+1)} \,, \qquad \forall\, l \in \mathrm{L} \qquad \forall\, p \in \mathrm{P} \qquad (6)$$

Para se avaliar o cumprimento da regra FIFO, um dos três KPI's estudados neste projeto, desenvolveram-se três restrições descritas de seguida. Na primeira restrição deste conjunto (7), procurou-se guardar na variável $FIFO_{ij}$, o valor de FIFO associado ao PC i alocado à posição j da sequência de extração.

$$FIFO_{ij} = X_{ij} \times valorFIFO_i\,, \qquad \forall\, i \in \mathrm{N} \qquad \forall\, j \in \mathrm{S} \qquad (7)$$

Na segunda restrição deste conjunto (8), procurou-se garantir que uma nova variável binária, denominada $JFIFO_j$, assumia obrigatoriamente o valor de 0 no caso do valor de FIFO esperado na posição j ser diferente do valor de FIFO do PC i alocado. Desta forma, a variável $JFIFO_j$ apenas pode assumir o valor de 1, caso o valor de FIFO esperado na posição j seja igual ao valor de FIFO do PC i alocado a essa posição.

$$seqFIFO_j \times X_{ij} \neq FIFO_{ij} \Longrightarrow JFIFO_j = 0\,, \qquad \forall\, i \in N \qquad \forall\, j \in \mathrm{S} \qquad (8)$$

Na restrição (9), terceira deste conjunto, calculou-se a percentagem de posições da sequência de extração que cumpriam a regra FIFO.

$$percFIFO\ = \frac{\sum_{j=1}^{S} JFIFO_j}{N\_seq} \qquad (9)$$

Para estudar o número exato de PC movimentados, segundo KPI estudado neste projeto, desenvolveram-se três restrições.

A primeira (10) permitiu guardar na variável Y_{lp} a posição do PC i caso este fosse alocado a uma qualquer posição j e 0 no caso contrário.

$$\sum_{j=1}^{S} X_{ij} \times POS_i = Y_{(l)(p)}\,, \qquad \forall\, i \in \mathrm{N} \qquad (10)$$

A segunda restrição deste conjunto (11) foi construída com o intuito de armazenar na variável MOV_l o valor da posição mais alta movimentada na linha l. Para esse efeito igualou-se esta variável no índice l ao valor máximo da variável Y_{lp} para cada linha l.

$$MOV_l = MAX\left(Y_{lp} \quad \forall\, p \in \mathrm{P}\right), \qquad \forall\, l \in \mathrm{L} \tag{11}$$

Por fim, na última restrição deste conjunto (12), guardou-se na variável $totMOV$ o número total de PC movimentados. Assim, utilizou-se o somatório da variável MOV_l, para todas as linhas existentes no armazém.

$$totMOV = \sum_{l=1}^{L} MOV_l \tag{12}$$

As últimas restrições desenvolvidas neste modelo matemático foram relativas ao estudo da medida de desempenho do tempo total de extração. No total construíram-se seis restrições para avaliar este KPI, divididas em três etapas que serão descritas de seguida.

Na primeira etapa desenvolveram-se duas restrições adequadas ao estudo do tempo de posição dos PC. A primeira restrição (13) permitiu guardar na variável W_{lp} o tempo de extração da posição do PC i nos índices l, linha de i e p, posição de i, caso este fosse alocado a uma qualquer posição j e 0 no caso contrário.

$$\sum_{j=1}^{S} X_{ij} \times tempExtrPOS_i = W_{(l)(p)}, \qquad \forall\, i \in \mathrm{N} \tag{13}$$

Na segunda restrição deste conjunto (14) armazenou-se na variável $tempPOS_l$ o valor do tempo de extração mais elevado em cada linha l. Para esse efeito igualou-se a variável $tempPOS_l$, no índice l ao valor máximo da variável W_{lp} para essa linha l.

$$tempPOS_l = MAX\left(W_{lp} \quad \forall\, p \in P\right), \qquad \forall\, l \in \mathrm{L} \tag{14}$$

Na segunda etapa avaliou-se o tempo de linha na qual o PC se encontrava. Assim, na primeira restrição (15) assegurou-se que sempre que a variável W_{lp} assumisse um valor diferente de 0, significando isto que o PC da linha *l* e da posição *p* tinha sido selecionado, atribuir-se-ia à variável binária $auxTempLIN_{lp}$ o valor de 1.

$$W_{lp} \neq 0 \Longrightarrow auxTempLIN_{lp} = 1\,, \qquad \forall\, l \in L \qquad \forall\, p \in \mathrm{P} \qquad (15)$$

A segunda restrição desta etapa (16) permitiu armazenar na variável $tempLIN_l$ o somatório dos tempos de extração de todos os PC selecionados em cada linha l associado ao percurso desde a linha l até à linha mais próxima da zona de *output*.

$$tempLIN_l = \sum_{p=1}^{P} auxTempLIN_{lp} \times tempExtrLIN_l\,, \quad \forall\, l \in L \quad (16)$$

Por fim, na terceira etapa contabilizou-se o tempo total de extração a partir das variáveis definidas acima. Na primeira restrição (17) somaram-se, para cada linha l os tempos de extração das posições e das linhas de armazenamento, guardados nas variáveis $tempPOS_l$ e $tempLIN_l$, respetivamente.

$$totLIN_l = tempPOS_l + tempLIN_l\,, \qquad \forall\, l \in L \qquad (17)$$

Por fim, na última restrição desenvolvida (18), totalizou-se o tempo total de extração através do somatório de todas as variáveis $totLIN_l$, definidas anteriormente.

$$totTEMPO = \sum_{l=1}^{L} totLIN_l \qquad (18)$$

4.6 Linguagem Python e Ambiente de Desenvolvimento Integrado PyCharm

Para resolver o problema de otimização modelado na secção anterior optou-se pela linguagem de programação Python na sua versão 3.7. Esta linguagem, desenvolvida em 1991 por Guido van Rossum, é de acesso gratuito, fácil interpretar e utilizar, possui uma vasta documentação e uma elevada variedade de bibliotecas desenvolvidas e fornecidas pela sua comunidade (Python Software Foundation, 2020). Uma das principais características da linguagem Python, e que a torna mais fácil de utilizar que as linguagens tradicionais, é a sua legibilidade. Esta característica deve ao facto desta linguagem ter sido desenvolvida com especial enfoque na facilidade de interpretação dos programadores. Um outro aspeto relevante desta linguagem orientada a objetos é o aumento de produtividade do programador em comparação com as linguagens C, C++ e Java. Esta propriedade prende-se sobretudo com facto do código em Python ser tipicamente, entre um-terço e um-quinto, do tamanho do código das linguagens supracitadas, aumentando assim a eficiência e reduzindo os erros de código. Da mesma forma, a linguagem Python tem a capacidade de integrar diversos componentes, como bibliotecas das linguagens C e C++ e pode, por sua vez, integrar componentes JAVA e .NET (Lutz, 2009). Para o autor Lutz (2009), que trabalhou com a linguagem Python mais de 20 anos, a principal desvantagem desta linguagem prende-se com a menor velocidade de execução do código em comparação com as linguagens C e C++. Um programa desenvolvido em Python é compilado a partir da fonte para um formato intermediário, conhecido como código de *bytes*, e posteriormente interpretado. Esta funcionalidade permite que o programa desenvolvido seja amplamente usado, uma vez que, o código de *bytes* é independente da plataforma usada. No entanto, o facto desta linguagem de programação não ser compilada diretamente para código binário, executado pelos processadores dos computadores, torna a sua execução mais demorada. Apesar deste aspeto, Lutz (2009) defende que a arquitetura, a simplicidade da sintaxe, o número de bibliotecas disponíveis, entre outros, permitem que um programa seja desenvolvido de forma mais rápida e eficiente do que as restantes linguagens. Desta forma, o Python compensa em velocidade de desenvolvimento a sua menor velocidade de execução. Por todas estas razões a popularidade e a utilização da linguagem de programação Python tem aumentado largamente nos últimos anos (Raschka & Mirjalili, 2019).

Com o intuito de facilitar o desenvolvimento do código na linguagem Python, selecionou-se o ambiente de desenvolvimento integrado, criado e disponibilizado pela JetBrains s.r.o., PyCharm na versão 2020.1. A empresa lançou duas versões, uma de acesso livre, disponível para toda a comunidade, e outra, utilizada neste projeto, com maior número de funcionalidades que se encontra disponível gratuitamente para fins académicos. A utilização do ambiente de desenvolvimento PyCharm proporciona

diversas vantagens, tais como: análise e deteção de erros de código, identificação de comandos e funções da linguagem, identificação inteligente e sugestão de código, exploração rápida dos documentos existentes no projeto a desenvolver, incorporação de diferentes bibliotecas no projeto, desenvolvimento, teste e execução de programas, entre outras (JetBrains, 2020). O PyCharm permitiu, pelas razões descritas acima, um desenvolvimento mais eficiente do modelo matemático na linguagem de programação Python.

4.7 Desenvolvimento de Algoritmos em Linguagem Python

Com o modelo matemático e as instâncias devidamente definidas procurou-se implementar em linguagem Python os algoritmos necessários para dar resposta ao problema real identificado neste projeto de dissertação.

O primeiro passo consistiu na criação de um novo projeto no ambiente de desenvolvimento integrado PyCharm. Posteriormente, procedeu-se à importação dos dados contidos nos documentos Excel através da biblioteca Pandas. Esta biblioteca foi desenvolvida e disponibilizada de forma gratuita para facilitar a manipulação e análise de dados em Python. O Anexo I presente na página 99 descreve de forma detalhada toda a implementação em Python do processo de importação de dados com auxílio da biblioteca Pandas.

Com os dados disponíveis no PyCharm realizou-se uma investigação sobre as bibliotecas disponibilizadas pela comunidade Python que auxiliam na implementação de modelos de programação linear inteira mista. De acordo com a informação recolhida optou-se pela modelação com auxílio da biblioteca Pulp. Este módulo foi escrito em linguagem Python e disponibilizado de forma gratuita para resolver problemas de otimização combinatória a partir de modelos matemáticos. A documentação do Pulp permite que a sua implementação seja fácil e acessível a programadores com e sem experiência (Pulp Documentation Team, 2009). Após a instalação do Pulp no computador pessoal, declararam-se os dados nas estruturas requeridas por esta biblioteca. De seguida, com auxílio das funções do Pulp implementaram-se a função objetivo e as três primeiras restrições do modelo matemático definido na secção 4.5. Ao estudar a documentação da biblioteca Pulp percebeu-se que esta não possuía uma função para a implementação direta da expressão "*implica*" definida na restrição (5) da secção 4.5. Esta limitação obrigaria a que a implementação do modelo em Python fosse mais complexa e exigisse maior capacidade computacional. Por esta razão, optou-se por terminar neste ponto a implementação do

problema de otimização combinatória através da biblioteca Pulp. A implementação do modelo construído com a biblioteca Pulp encontra-se descrita detalhadamente no Anexo II presente na página 103.

Após uma nova investigação, optou-se por utilizar a biblioteca Pyomo (*Python Optimization Modeling Objects*). Esta biblioteca é constituída por um conjunto de módulos com diversas funções para formulação em Python de problemas de otimização combinatória, passiveis de modelação matemática. O Pyomo incorpora as principais características das linguagens AML (*Algebraic Modeling Languages*) modernas, complementadas com a linguagem orientada a objetos Python (Hart, Laird, Watson, & Woodruff, 2017). Através da documentação do Pyomo, em especial do livro dos autores Hart et al. (2017), perceberam-se os passos para a instalação e os requisitos de utilização desta biblioteca. De forma a tirar proveito da importação de dados desenvolvida anteriormente, optou-se por declarar um modelo concreto com os dados instanciados diretamente no modelo. Com as estruturas de dados no formato adequado implementaram-se a função objetivo e as três primeiras restrições do modelo matemático definido, com auxílio das funções da biblioteca Pyomo. Contudo, a documentação do Pyomo não contempla uma função para a implementação direta da expressão "*implica*" definida na restrição (5) da secção 4.5. Esta limitação obrigaria a que a implementação em Python fosse mais complexa e exigisse maior capacidade computacional para resolver o modelo. Deste modo, e de forma análoga ao realizado com a biblioteca Pulp, optou-se por terminar neste ponto a implementação do problema de otimização combinatória através da biblioteca Pyomo. O Anexo III presente na página 108 desta dissertação apresenta de forma detalhada a implementação do modelo realizada com a biblioteca Pyomo.

A terceira biblioteca utilizada para implementar o modelo matemático desenvolvido foi o Docplex, disponibilizada de acesso livre pela IBM para a modelação e otimização em Python, através do *solver* CPLEX. Esta biblioteca permite resolver o problema modelado num computador pessoal, com a instalação prévia da versão IBM ILOG CPLEX Optimization Studio V12.8, ou versões mais recentes, ou, em alternativa, no serviço *cloud* da própria IBM (Python Software Foundation, 2020). Neste projeto de dissertação optou-se por resolver no computador pessoal. Após uma investigação inicial à documentação da biblioteca Docplex disponível na plataforma GitHub percebeu-se que a versão comunitária do *solver* CPLEX disponível na biblioteca Docplex, não seria suficiente para resolver uma instância com o volume de dados existente neste projeto (IBM, 2020). Por esta razão, e de acordo com informação recolhida, instalou-se na própria biblioteca Docplex, a versão do *solver* CPLEX disponível no IBM ILOG CPLEX Optimization Studio V12.8. O acesso a esta versão é disponibilizado gratuitamente pela IBM para fins

académicos. Este passo foi essencial para que, o *solver* utilizado não estivesse limitado e pudesse resolver as instâncias submetidas.

Com a biblioteca devidamente instalada no computador pessoal definiram-se os dados nas estruturas adequadas e requeridas pelo Docplex. De seguida, iniciou-se a construção do modelo matemático definido na secção 4.5. Através das funções do Docplex declararam-se as variáveis necessárias ao processo de otimização. Posteriormente, definiu-se a função objetivo e as 6 primeiras restrições do modelo matemático. Com este conjunto de passos produziu-se o modelo MPA (Modelo Problema de Afetação) que é similar ao problema de afetação e permitiu implementar e testar o modelo e as instâncias geradas. Este modelo não permite avaliar diretamente nenhuma medida de desempenho estudada, durante o processo de otimização.

Com o intuito de avaliar a medida de desempenho relativa à percentagem de FIFO cumprida, implementaram-se as restrições (7), (8) e (9) apresentadas na secção 4.5. Com este conjunto de restrições adicionadas ao modelo MPA previamente definido produziu-se o modelo MF (Modelo FIFO). Este modelo permite avaliar e condicionar de forma direta a percentagem de FIFO cumprida durante o processo de otimização.

De forma a incorporar os restantes KPI's estudados neste projeto, implementaram-se as últimas 9 restrições do modelo matemático definido na secção 4.5. As primeiras três restrições permitiram avaliar diretamente no processo de otimização o número exato de PC movimentados. Já as seis últimas restrições implementadas possibilitaram estudar a medida de desempenho associada ao tempo total de extração. Com a associação destas restrições aos modelos MPA e MF produziu-se o modelo mais completo MTFM (Modelo Tempo FIFO Movimentados). É importante referir que, apesar das vantagens da incorporação dos 3 KPI's diretamente no modelo de otimização, o aumento do número de variáveis e restrições tornou o problema mais difícil de resolver e, por isso, mais exigente computacionalmente o que, em determinadas situações pode inviabilizar a obtenção de uma solução ótima em tempo útil.

Para que fosse possível avaliar os valores obtidos nas três medidas de desempenho estudas independentemente do modelo utilizado, MPA, MF ou MTFM, foi implementado um outro algoritmo. Através de uma função do Docplex armazenaram-se os valores da solução ótima obtida para a variável X_{ij} numa estrutura de dados do tipo dicionário. Com estes valores já otimizados desenvolveram-se três conjuntos de funções em linguagem de programação Python que permitiram calcular os valores dos KPI's: tempo total de extração, percentagem de FIFO cumprida e número de PC movimentados. Estes valores foram obtidos a partir da solução ótima e não durante o processo de otimização pelo que correspondem aos valores reais destas medidas de desempenho para a solução ótima obtida.

Por fim, desenvolveu-se em Python um algoritmo que permitiu visualizar graficamente o Shopstocker com auxílio da biblioteca Matplotlib. Este módulo possui um conjunto de funções que permitem criar visualizações estáticas, dinâmicas ou interativas em linguagem Python. O estudo da documentação do Matplotlib, disponibilizada por Hunter, Dale, Firing, Droettboom, & Team (2012), permitiu inferir quais as funções e parâmetros que melhor se enquadravam neste projeto. Este último algoritmo desenvolvido tira igualmente proveito do dicionário com os valores otimizados da variável X_{ij} para construir uma representação gráfica do Shopstocker. Esta perspetiva gráfica permite observar de forma imediata a localização dos PC selecionados ao longo do Shopstocker.

Em suma, com os algoritmos desenvolvidos é possível obter uma solução ótima para o problema real em estudo neste projeto, é possível avaliar diretamente os valores dos três KPI's, isto é, não existe a necessidade de cálculos posteriores à execução dos algoritmos, e consegue-se visualizar graficamente a solução obtida. Assim, com a modelação em linguagem Python descrita nesta secção, puderam-se realizar as experiências computacionais apresentadas no capítulo seguinte. A implementação dos modelos, MPA, MF, MTFM e dos restantes algoritmos realizada com auxílio das bibliotecas Pandas, Docplex e Matplotlib encontra-se descrita de forma detalhada no Anexo IV presente na página 114.

5. EXPERIÊNCIAS COMPUTACIONAIS

Neste capítulo são descritas as experiências computacionais realizadas com os modelos desenvolvidos e apresentados no capítulo anterior. Em cada subsecção encontram-se expostas as condições nas quais as experiências foram realizadas, assim como dos resultados obtidos para as três medidas de desempenho analisadas: tempo de extração, percentagem de FIFO e número de PC movimentados. No final são discutidas as principais causas identificadas para os resultados observados.

5.1 Experiências de Validação

Numa primeira fase, realizaram-se experiências de validação utilizando o modelo MTFM desenvolvido na linguagem de programação Python. Uma vez que este modelo é o mais completo e incorpora as restrições dos restantes, a sua validação aprova igualmente os restantes modelos. De forma a validar este modelo criou-se uma sequência de teste com 15 posições de extração. Esta experiência teve como objetivo validar as variáveis e restrições desenvolvidas relativas aos KPI's, tempo de extração, percentagem de FIFO cumprida e PC movimentados por extração. Assim, e de acordo com estes objetivos, importaram-se para o modelo os dados relativos à configuração 1 e à sequência de teste com 15 posições de extração e realizaram-se as experiências com recurso a três funções objetivo distintas:

a) minimização do tempo de extração corrigido;
b) minimização do valor complementar da percentagem de FIFO cumprida;
c) minimização do número de PC movimentados.

A Tabela 7 apresenta os resultados obtidos nas três experiências de validação, onde se avaliou individualmente o funcionamento das restrições desenvolvidas para cada KPI.

Tabela 7 - Resultados das experiências de validação.

Experiência	KPI		
	tempo (s)	FIFO (%)	mov (uni)
1	891	53,33	12
2	1621	100,00	47
3	1735	53,33	0

A referida tabela apresenta, para cada experiência, o tempo total de extração medido em segundos, a percentagem de FIFO cumprida e o número de PC movimentados por impedimento físico de acesso aos 15 PC da sequência.

5.1.1 Experiência de Validação 1: Tempo de Extração

Nesta experiência o modelo de programação linear inteira mista apenas procurou minimizar o tempo de extração dos 15 PC. A Figura 22 ilustra graficamente a solução obtida.

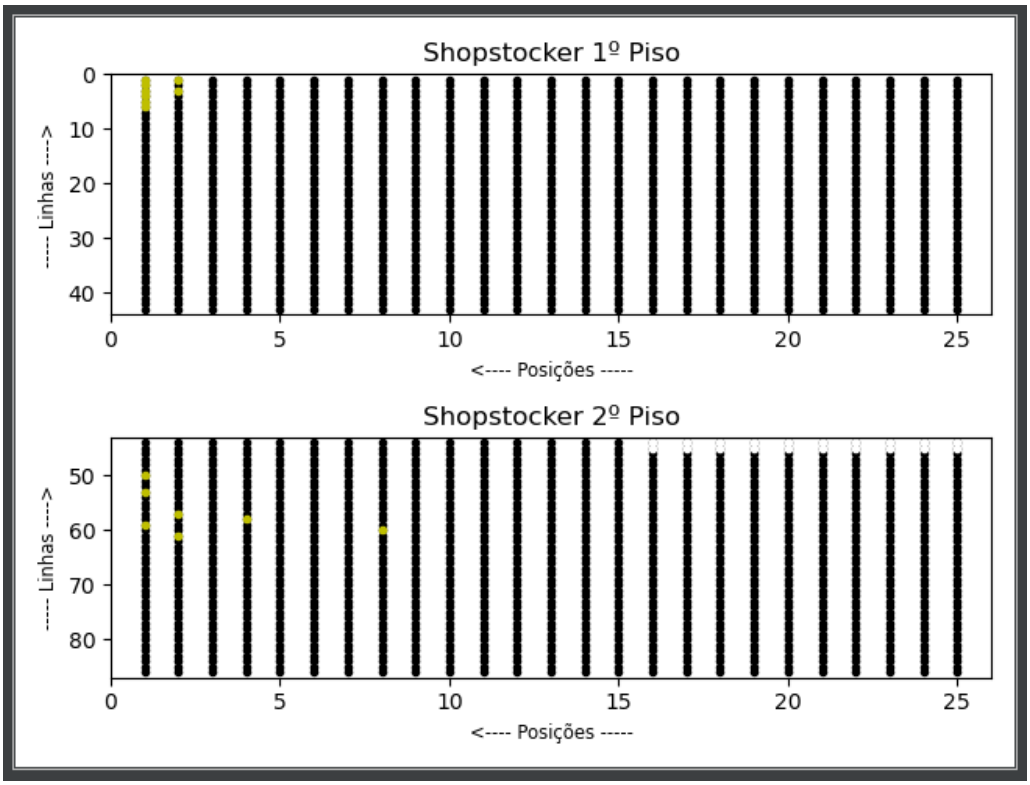

Figura 22 - Apresentação gráfica dos PC selecionados na experiência de validação 1.

Foram selecionados os PC cujo somatório dos tempos de extração era mínimo, por se encontrarem mais perto da zona de *output*. Este resultado valida que o modelo é capaz de identificar os PC com menor tempo de extração e cumpre as regras de funcionamento do sistema real, nomeadamente a regra de extração prévia de todos os PC posicionados à frente de um PC escolhido para a sequência.

A Tabela 7 informa que, para extrair os 15 PC requeridos na sequência de extração foi necessário um tempo de 891 segundos. Uma vez que a solução obtida se trata da solução ótima, então sabe-se que não é possível para este conjunto de dados obter um tempo de extração inferior a este. Deste modo é encontrada a situação limite para o tempo de extração desta sequência, na configuração 1 do armazém. Este tempo associado à situação limite do tempo de extração é a referência temporal para esta instância.

Por outro lado, como a função objetivo não valorizava o efeito da percentagem de FIFO cumprida nem o número de PC movimentados, a solução obtida não teve preocupações com estes dois KPI's. Na

Tabela 7 verifica-se que se movimentaram 12 PC a mais por impedimento físico, e apenas 53,33% das 15 posições de extração cumpriram a regra FIFO.

5.1.2 Experiência de Validação 2: Percentagem de FIFO

Nesta experiência procurou-se validar as restrições desenvolvidas com o intuito de avaliar percentagem de FIFO cumprida, que o modelo consegue alcançar. Por esta razão, a função objetivo considerada foi a minimização do valor complementar da percentagem de FIFO cumprida. A Figura 23 ilustra graficamente a solução obtida.

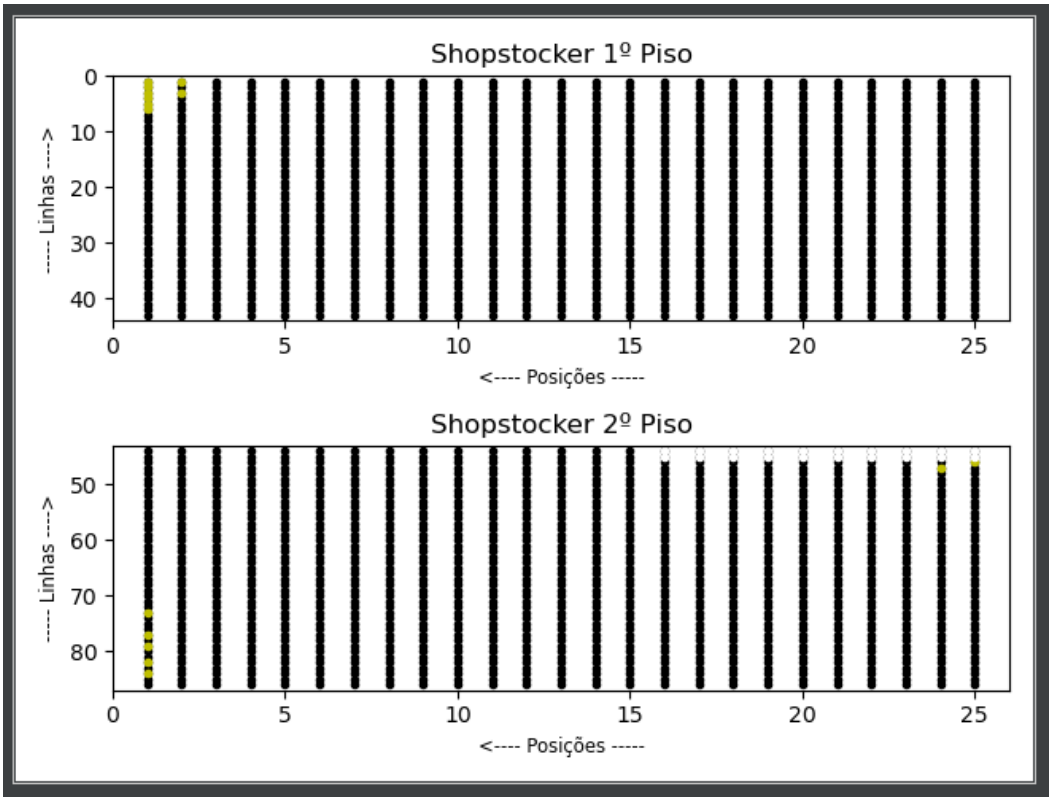

Figura 23 - Apresentação gráfica dos PC selecionados na experiência de validação 2.

A solução ótima obtida e apresentada na Tabela 7 indica que foi possível atingir a percentagem máxima de FIFO (100%) para a instância submetida, configuração 1 e sequência de teste de 15 posições. Este resultado valida que o modelo é capaz de alcançar o máximo do valor de FIFO, cumprindo as regras do sistema real, e satisfazendo a sequência solicitada.

A subida na percentagem de FIFO cumprida face à experiência anterior teve uma consequência clara no aumento do tempo total de extração e do número de PC movimentados. Tal como a Figura 23 ilustra, os PC foram selecionados apenas com base no seu valor de FIFO. Numa análise mais profunda

à solução ótima obtida, destacam-se os casos extremos de dois PC terem sido extraídos das posições 24 e 25 das suas linhas de armazenamento, por terem o valor de FIFO adequado, obrigando a um aumento considerável de PC movimentados a mais para além dos 15 solicitados na sequência, passando-se de 12 para 47 PC movimentados em excesso.

Concomitantemente, o aumento da profundidade das posições selecionadas e a escolha de linhas mais afastadas da zona de *output*, tem um efeito claro no aumento do tempo de extração dos PC necessários para responder à sequência pedida, prejudicando a taxa de extração do sistema.

5.1.3 Experiência de Validação 3: Número de PC Movimentados

Na última experiência de validação realizada, procurou-se confirmar as restrições desenvolvidas para o cálculo do número de PC movimentados para responder a uma sequência de extração. Para tal, desenvolveu-se a função objetivo de minimização do número de PC movimentados. A Figura 24 apresenta graficamente os PC selecionados na solução ótima desta experiência.

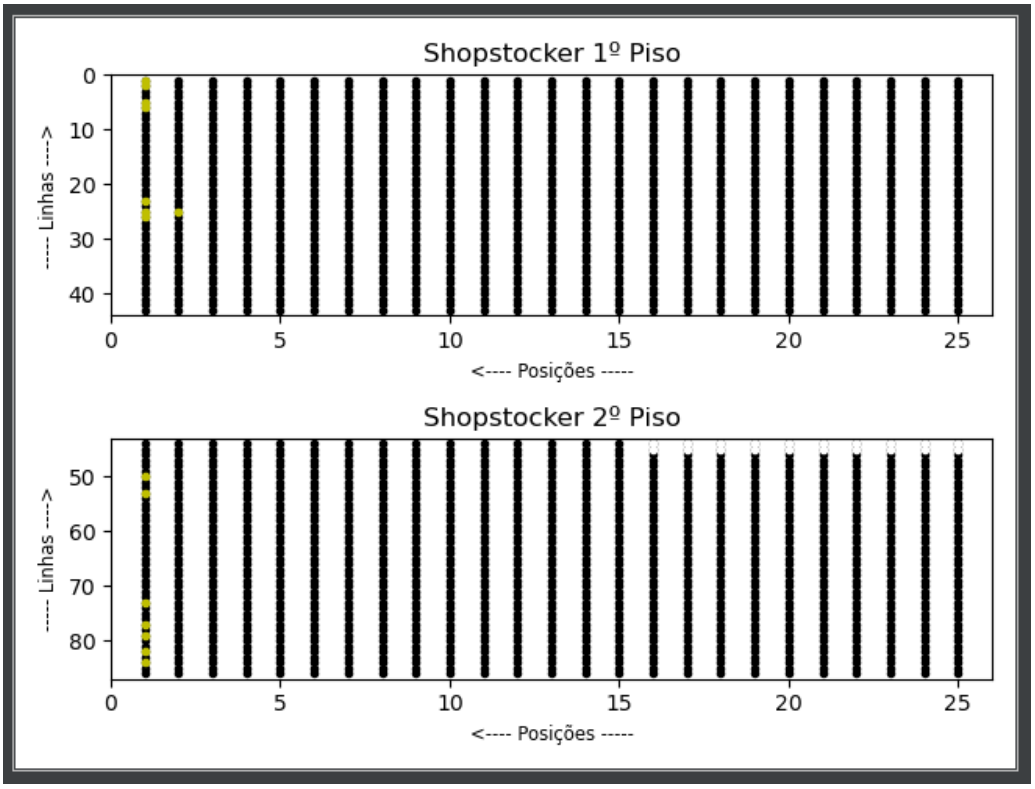

Figura 24 - Apresentação gráfica dos PC selecionados na experiência de validação 3.

Como é possível verificar na figura, o modelo selecionou os PC armazenados nas posições mais baixas das linhas existentes no Shopstocker. Como não existe contabilização de FIFO nem de tempo de

extração, a solução obtida procurou minimizar o número de PC movimentados que respondessem à sequência pretendida. Por esta razão, e tal como a Tabela 7 permite perceber, na solução obtida não houve qualquer PC movimentado a mais para além da extração dos 15 da sequência. A análise mais detalhada da solução, confirma que todos os PC escolhidos posicionados na segunda posição de uma linha, tiveram o PC armazenado à sua frente a ser também movimentado para a sequência solicitada, num instante anterior. Esta experiência valida a situação de que é possível obter a solução nula em termos de movimentação em excesso de PC que não formam a sequência, se os escolhidos estiverem em posições adequadas nas linhas. Por outro lado, como a função objetivo do modelo de programação linear inteira mista não considerou o tempo nem a percentagem de FIFO cumprida, a solução obtida não obteve valores interessantes para esses KPI's. Assim, obteve-se um valor de apenas 53,33% de cumprimento de FIFO e um tempo de extração de 1735 segundos, que contrasta com o valor 891 obtido na otimização do tempo de extração.

Com estas três experiências, foi possível validar o modelo MTFM, uma vez que, cada KPI minimizou o seu valor quando submetido de forma exclusiva na função objetivo. O modelo cumpre por isso os pressupostos pelo qual foi desenvolvido.

5.2 Modelo MF – Configuração 1 e Configuração 2

No primeiro conjunto de experiências realizadas com o modelo MF submeteram-se 30 sequências de extração de PC ao Shopstocker, considerando este no estado da configuração 1. Na função objetivo deste modelo foram atribuídos diferentes pesos às parcelas consideradas, procurando-se avaliar a relação entre a percentagem de FIFO cumprida, o tempo de extração e o número de PC movimentados. A escolha destas ponderações procurou explorar um espaço de soluções ótimas no sentido de se conhecer o conjunto de soluções mais promissoras, através da alteração sucessiva do peso atribuído à percentagem de FIFO cumprida e ao parâmetro do tempo nominal de extração. As oito ponderações atribuídas nas experiências computacionais realizadas foram:

1. 100% FIFO + 0% tempo nominal de extração;
2. 75% FIFO + 25% tempo nominal de extração;
3. 60% FIFO + 40% tempo nominal de extração;
4. 53% FIFO + 47% tempo nominal de extração;
5. 50% FIFO + 50% tempo nominal de extração;
6. 40% FIFO + 60% tempo nominal de extração;

7. 25% FIFO + 75% tempo nominal de extração;
8. 0% FIFO + 100% tempo nominal de extração.

A Tabela 8 apresenta para cada experiência, os valores registados para os três KPI's em estudo, obtidos no ambiente de desenvolvimento integrado PyCharm.

Tabela 8 - Resultados das experiências computacionais realizadas com a configuração 1.

F Obj	100% FIFO			75% F + 25% t			60% F + 40% t			53% F + 47% t			50% F + 50% t			40% F + 60% t			25% F + 75% t			100% t_ext		
SEQ	KPI			KPI			KPI			KPI			KPI			KPI			KPI			KPI		
	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov
1	8578	10	181	7957	10	134	7810	11	115	7166	16	102	7046	17	102	6928	18	101	6547	22	104	6371	24	82
4	7986	5	102	7969	5	103	7677	7	94	7599	8	98	7455	9	95	7161	12	82	6867	16	69	6835	22	75
9	8320	7	140	7904	7	83	7643	12	96	7613	11	76	7690	13	85	6923	19	46	6810	20	45	6536	21	42
16	6957	3	84	6865	3	70	6681	4	62	6554	6	58	6335	8	60	6102	12	49	5940	14	55	6452	43	54
21	7374	4	103	6966	4	67	6794	5	58	6671	6	57	6467	8	59	6379	9	58	6323	10	61	6195	13	70
31	6650	2	80	6527	2	64	6523	3	61	6286	5	57	6286	5	57	5971	9	52	5917	10	44	5884	11	48
32	6743	5	106	6479	5	78	6479	5	78	6265	7	65	6026	9	62	5597	16	54	5597	16	54	5364	19	63
36	6979	4	108	6730	4	75	6522	5	64	6402	6	64	6316	7	67	6064	10	58	5722	14	69	6125	39	65
37	7109	4	98	7011	4	72	7011	4	72	6631	7	67	6631	7	67	6456	9	67	6351	11	67	6141	15	72
49	7759	4	108	7635	4	85	7138	7	66	6616	11	62	6496	12	62	6144	14	68	6054	17	68	6720	45	80
55	7263	6	111	6820	6	50	6820	6	50	6408	10	56	6408	10	56	6180	13	50	5921	16	57	5811	19	57
65	5565	2	85	5295	2	50	5295	2	50	5182	3	49	5078	4	51	4901	6	57	4818	8	56	4950	14	60
71	6989	2	128	6776	2	102	6436	4	87	6324	5	88	6324	5	88	6203	7	81	6047	10	89	6864	40	88
74	6680	4	100	6569	4	88	6342	6	69	6223	7	66	6122	8	49	5903	11	61	5614	14	63	5579	16	58
86	7209	2	73	7143	2	76	6781	4	62	6604	6	68	6604	6	68	6219	10	68	5986	13	67	6178	46	76
91	7065	2	95	6875	2	70	6584	4	48	6174	8	53	6056	9	52	5908	12	56	5737	14	59	5717	17	54
94	7770	5	110	7251	5	57	7081	6	47	6971	7	47	6760	9	50	6666	10	52	6502	13	54	7045	36	80
101	6056	2	67	5850	2	45	5678	3	36	5559	4	38	5230	7	40	4984	10	43	4953	11	36	5960	54	55
103	7728	6	111	7281	6	67	7110	7	55	6895	8	45	6512	12	49	6391	14	57	6391	14	57	5982	20	49
106	7889	3	148	7547	3	104	6937	7	84	6571	10	77	6464	11	78	6093	16	36	6093	16	36	6395	42	50
111	7150	6	165	6803	6	126	6235	10	90	6000	12	90	6000	12	90	5742	15	94	5463	20	76	5996	50	82
113	6476	3	132	6074	3	78	6074	3	78	5961	4	77	5849	5	78	5640	8	90	5576	11	92	6187	32	99
115	7275	4	125	7066	4	97	6955	5	95	6318	10	71	6318	10	71	6045	13	75	5926	15	72	6639	40	73
116	6559	2	138	6108	2	96	5758	4	76	5377	7	54	5377	7	54	5100	10	60	5055	12	65	4955	14	60
123	7779	4	133	7391	5	77	7027	7	59	6710	10	60	6585	11	60	6406	13	62	6352	14	54	6327	15	59
127	8452	6	189	7705	6	105	7646	9	97	7009	13	83	7015	16	85	6839	18	88	6325	21	75	6243	24	81
133	7876	4	117	7543	4	76	7469	5	78	7186	7	72	6792	10	64	6557	13	59	6507	14	49	6361	17	57
145	6213	4	126	5983	4	101	5829	5	93	5829	5	93	5709	6	93	5174	12	58	4990	15	65	5424	42	68
149	7818	5	126	7394	5	88	6885	8	50	6623	10	46	6497	11	44	6119	15	48	5817	18	49	6575	38	55
153	7737	4	124	7635	4	110	7285	7	105	7173	10	81	7141	10	77	6718	16	76	6511	18	77	7058	46	71
min	5565	2	67	5295	2	45	5295	2	36	5182	3	38	5078	4	40	4901	6	36	4818	8	36	4950	11	42
média	7267	4,13	117	6972	4,17	83	6750	5,83	73	6497	7,97	67	6386	9,13	67	6117	12,3	64	5957	14,6	63	6162	29,1	66
max	8578	10	189	7969	10	134	7810	12	115	7613	16	102	7690	17	102	7161	19	101	6867	22	104	7058	54	99

Na primeira linha da tabela são identificadas as ponderações utilizadas na experiência, enquanto que a primeira coluna identifica a sequência de extração submetida ao modelo. Na Tabela 8 verifica-se ainda que para cada sequência foram realizadas 8 experiências, correspondentes às ponderações enunciadas acima, das quais se registaram os valores das medidas de desempenho utilizadas: *tempo*, correspondente ao tempo total de extração em segundos; *FIFO*, que representa o valor complementar da percentagem de FIFO cumprida; e *mov*, que corresponde ao número de PC movimentados a mais para além dos da sequência. As 3 últimas linhas da tabela apresentam o valor mínimo, médio e máximo da respetiva coluna. Na Tabela 8 são apresentados os tempos reais de extração calculados com um algoritmo a partir dos valores da solução obtida pelo modelo de otimização que considera os tempos nominais, isto é, que não considera o efeito de compensação existente no Shopstocker.

No segundo conjunto de experiências, submeteram-se as mesmas 30 sequências à configuração 2. As condições de realização destas experiências foram iguais às condições de realização das experiências com a configuração 1. Desta forma, para além da relação entre os diferentes KPI's avaliados, procurou-se observar as consequências da adoção da configuração 2, em detrimento da organização existente na configuração 1. Os resultados observados estão ilustrados na Tabela 9, onde se registaram os valores das medidas de desempenho, tempo total de extração, complementar da percentagem de FIFO cumprida e número total de PC movimentados.

Tabela 9 - Resultados das experiências computacionais realizadas com a configuração 2.

F Obj	100% FIFO			75% F + 25% t			60% F + 40% t			53% F + 47% t			50% F + 50% t			40% F + 60% t			25% F + 75% t			100% t_ext		
SEQ	KPI			KPI			KPI			KPI			KPI			KPI			KPI			KPI		
	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov	tempo	FIFO	mov
1	8386	7	197	8108	7	176	7991	8	167	7991	8	167	7779	11	133	7653	13	116	7573	15	106	7537	20	104
4	7843	6	121	7761	6	117	7761	6	117	7686	7	102	7686	7	102	7530	10	95	7487	13	104	7428	22	111
9	8232	5	129	8008	5	111	8006	5	117	7777	8	74	7810	9	80	7726	10	67	7558	14	66	7917	38	74
16	7879	6	193	7061	6	102	7061	6	102	6993	7	91	6891	8	72	6824	9	58	6790	10	65	6738	14	61
21	7192	1	104	7168	1	106	7168	1	106	7181	2	107	7135	3	105	7033	4	101	6938	6	96	7488	28	106
31	6682	3	89	6347	3	59	6347	3	59	6347	3	59	6347	3	59	6268	4	56	6153	6	46	6090	12	40
32	6912	4	119	6435	4	80	6435	4	80	6325	5	75	6325	5	75	6109	8	58	6000	11	45	6251	33	52
36	7261	5	132	6948	5	76	6948	5	76	6948	5	76	6926	6	72	6858	7	76	6792	9	84	6794	12	83
37	7294	6	143	6642	6	79	6489	7	58	6489	7	58	6385	8	50	6385	8	50	6221	12	42	6741	42	52
49	7289	5	108	7167	5	109	7167	5	109	7167	5	109	7167	5	109	6949	9	88	6911	11	82	6934	17	73
55	7726	4	157	7386	4	127	7321	5	127	7061	7	102	6951	8	82	6891	9	87	6810	11	80	6811	20	82
65	5409	0	73	5409	0	73	5243	1	61	5243	1	61	5151	2	42	5151	2	42	5086	3	42	5690	31	45
71	7122	4	139	6633	4	76	6633	4	76	6466	6	57	6466	6	57	6398	7	61	6355	8	55	6964	29	73
74	7398	6	176	6465	6	85	6465	6	85	6212	8	49	6212	8	49	6212	8	49	6094	11	48	6163	13	56
86	7578	4	151	7107	4	104	7107	4	104	7032	5	94	6930	6	75	6824	8	78	6824	8	78	8024	48	88
91	6925	6	120	6408	6	61	6408	6	61	6408	6	61	6408	6	61	6358	7	61	6290	9	70	6316	14	77
94	7946	1	147	7750	1	130	7750	1	130	7536	3	102	7466	4	97	7196	8	77	7091	10	82	7054	30	78
101	5659	1	48	5555	1	35	5555	1	35	5555	1	35	5555	1	35	5555	1	35	5457	3	39	6099	40	48
103	7515	3	120	7332	3	89	7332	4	94	7168	4	71	7133	6	66	7307	8	69	7018	8	51	7015	19	60
106	7398	5	121	7113	5	86	7113	5	86	7113	5	86	7012	6	69	6834	8	68	6591	15	62	7374	40	68
111	7449	6	228	6596	6	122	6596	6	122	6596	6	122	6596	6	122	6396	10	102	6310	13	95	7246	47	97
113	6491	3	157	6331	3	142	6331	3	142	6331	3	142	6331	3	142	6331	3	142	6149	7	138	6469	30	133
115	7108	1	116	7047	1	104	7047	1	104	7047	1	104	6962	2	89	6876	3	87	6719	7	78	7494	37	93
116	5880	3	100	5609	3	73	5609	3	73	5609	3	73	5609	3	73	5609	3	73	5619	8	78	5578	10	66
123	7663	6	151	7050	6	90	7050	6	90	7050	6	90	6963	7	86	6861	8	82	6826	9	77	7002	29	94
127	8628	8	231	7669	9	148	7636	11	127	7518	9	126	7636	11	127	7636	11	127	7431	81	102	7183	23	101
133	7607	2	99	7331	2	77	7331	2	77	7311	3	72	7311	3	72	7311	3	72	7173	7	56	7199	15	63
145	6284	3	133	5886	3	97	5886	3	97	5886	3	97	5886	3	97	5840	6	85	5634	11	68	6405	40	70
149	7832	2	144	7646	2	127	7495	3	110	7243	5	71	7141	6	57	6985	8	65	6848	12	51	7462	40	59
153	8092	4	174	7699	4	113	7699	4	113	7625	5	120	7625	5	120	7569	7	113	7340	13	105	7685	46	125
min	5409	0	48	5409	0	35	5243	1	35	5243	1	35	5151	1	35	5151	1	35	5086	3	39	5578	10	40
média	7289	4	137	6922	4,03	99	6899	4,3	97	6830	4,9	88	6793	5,57	83	6716	7	78	6603	12	73	6905	28	78
max	8628	8	231	8108	9	176	8006	11	167	7991	9	167	7810	11	142	7726	13	142	7573	81	138	8024	48	133

A estrutura desta tabela é igual à Tabela 8, onde a primeira linha identifica as ponderações utilizadas e a primeira coluna a sequência de extração submetida. Para cada sequência foram realizadas oito experiências com diferentes ponderações atribuídas aos componentes da função objetivo e registaram-se os valores de três KPI's: *tempo*, que representa o tempo total de extração em segundos; *FIFO*, correspondente ao valor complementar da percentagem de FIFO cumprida; e *mov*, que representa o número de PC extra movimentados, para além dos requeridos na sequência. As 3 últimas linhas da tabela apresentam o valor mínimo, médio e máximo da respetiva coluna. Uma vez mais, os valores

apresentados na Tabela 9 para os tempos reais de extração foram calculados com um algoritmo a partir dos valores da solução obtida pelo modelo de otimização que considera os tempos nominais, isto é, que não considera o efeito de compensação existente no Shopstocker.

O modelo MF apresentou tempos médios de CPU de 8 segundos para resolver o problema e as instâncias submetidas.

Os resultados das duas tabelas apresentadas nesta subsecção foram compilados no gráfico exposto na Figura 25, onde se expõem as variações percentuais das medidas de desempenho obtidas com a configuração 2, comparando com a configuração 1.

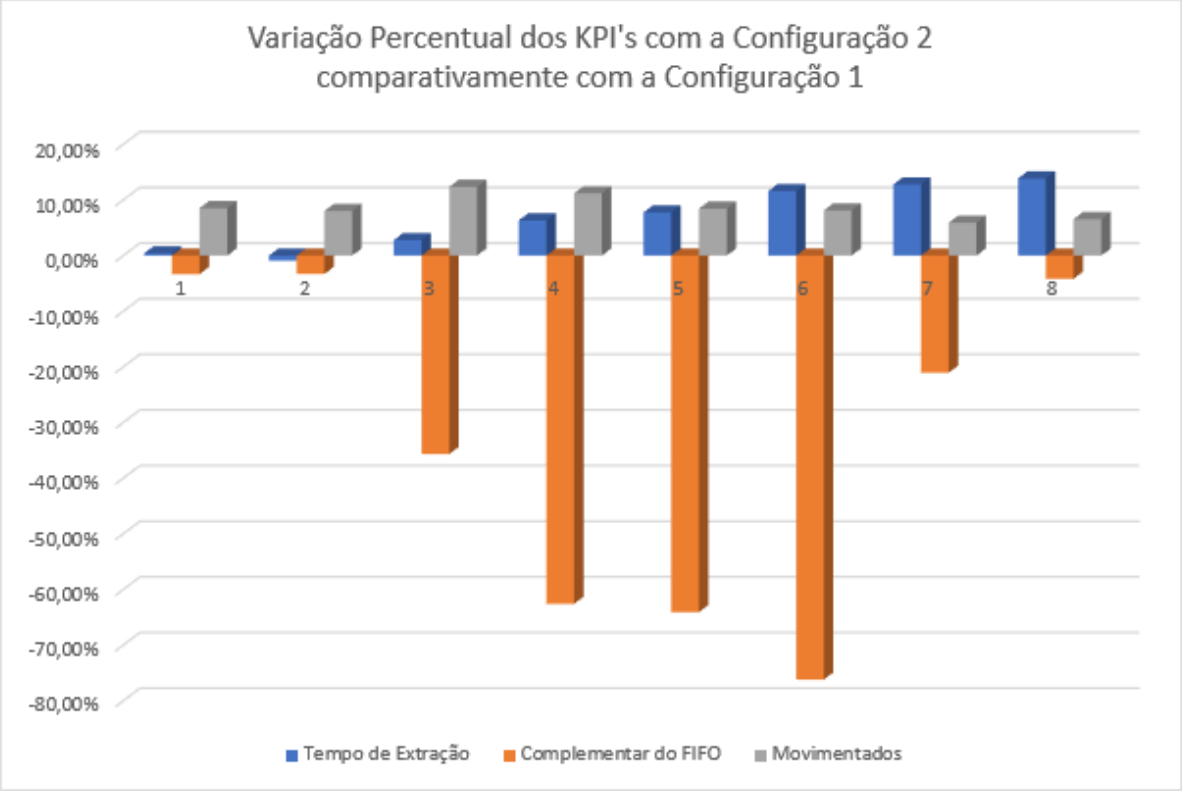

Figura 25 - Análise comparativa das configurações 1 e 2.

É importante referir que, a variação positiva nos KPI's do tempo total de extração (identificada a azul no gráfico) e do número de PC movimentados (identificada a cinzento no gráfico) significa que houve um agravamento no valor dessas medidas de desempenho com a adoção da configuração 2. Por outro lado, a variação negativa no valor complementar da percentagem de FIFO cumprida (identificada a laranja no gráfico) significa uma melhoria do valor desse KPI com a adoção da configuração 2 em detrimento da configuração 1.

De acordo com os resultados obtidos a adoção da configuração 2 resulta numa melhoria de 33,8% no valor complementar da percentagem de FIFO cumprida. Este padrão é ainda mais acentuado nas experiências 4, 5 e 6, ilustradas a laranja na Figura 25, onde o valor complementar da percentagem de FIFO diminui 62,6%, 64,1%, 76,2%, respetivamente, com a adoção da configuração 2.

Por outro lado, os resultados permitiram ainda depreender que, o comportamento das restantes medidas de desempenho piora com a adoção desta segunda configuração. O tempo total de extração aumenta significativamente, chegando a atingir aumentos médios de 13,87% comparativamente com a configuração 1. Da mesma forma, a configuração 2 resulta num aumento significativo do número de PC movimentados. Este aumento atinge os valores médios de 12,36% e 11,20% nas experiências 3 e 4, respetivamente, como se pode observar no gráfico presente na Figura 25.

Os resultados permitiram ainda inferir que o valor médio da diminuição do complementar da percentagem de FIFO, 33,8%, foi sistematicamente mais elevado que os aumentos percentuais do tempo total de extração, 6,82%, e do número de PC movimentados, 8,64%.

Através deste estudo realizado com as configurações representativas do estado do ShopStocker verificou-se uma melhoria na percentagem de FIFO cumprida com a adoção da configuração 2. Este facto é explicado pela distribuição dos PC das classes *medium* e *low runners* por um maior número de linhas de armazenamento, o que implicou uma diminuição da mistura de referências por linha. Desta forma, um PC extraído de uma linha caótica obriga à movimentação de mais suportes vazios e menos PC. Por esta razão, a probabilidade de se perder um PC da ordem FIFO reduz-se com a configuração 2, permitindo que mais PC estejam disponíveis para dar resposta à sequência requerida. Assim, com a maior disponibilidade de PC nas linhas de armazenamento é possível selecionar o PC correto para aumentar o cumprimento da regra FIFO.

Por outro lado, a dispersão dos PC por linhas mais afastadas da zona de output tem um efeito negativo no tempo total de extração. Este facto é explicado por dois fatores: percurso de saída mais longo e perda do efeito de compensação. Tal como no sistema estudado por Zaerpour et al. (2015), a ocupação de linhas mais afastadas da zona de saída obriga a que os PC selecionados tenham que percorrer uma distância maior, resultando num aumento do tempo de extração. Para além disso, existe uma perda do efeito de compensação em que a extração de um PC da posição p da linha A compensa a extração de um outro PC da linha A, numa posição superior a p. Como os PC estão espalhados por um maior número de linhas, a probabilidade de um PC compensar a saída de um segundo é menor. Por esta razão, existe um aumento do tempo total de extração com a configuração 2.

De igual modo, a perda do efeito de compensação resultou num maior número de PC movimentados com a configuração 2. Se os PC estiverem mais condensados, isto é, se forem armazenados num menor número de linhas, a extração de um PC da posição p da linha A irá aproximar da extremidade de saída todos os PC dessa linha em posições superiores a p. Se um segundo PC da linha A for selecionado, obrigará à movimentação de um menor número de PC, uma vez que a sua saída foi compensada pela extração do PC da posição p. Por outro lado, a dispersão dos PC por um maior número de linhas de armazenamento diminui a probabilidade dos PC selecionados serem da mesma linha, perdendo-se o efeito de compensação e, consequentemente, o número de PC movimentados aumenta.

5.3 Modelo MF – Análise Multiobjetivo

Marler & Arora (2004) definiram otimização multiobjectivo como sendo o processo de otimização sistemática e simultânea de vários objetivos. Segundo os autores, habitualmente não é possível encontrar uma solução ótima global para os diversos objetivos em estudo. Por esta razão, é necessário determinar um conjunto de pontos que se enquadrem com uma definição de ótimo. Geralmente, o conceito utilizado é o ótimo de Pareto segundo o qual um ponto é ótimo se não existir, no espaço de soluções, um ponto que o domine. Um ponto domina outro se, para o conjunto de todas as funções objetivo, consegue melhorar o valor de pelo menos uma função, sem piorar o valor das restantes. O conjunto de todos os pontos correspondentes ao ótimo de Pareto, não dominados, produzem a curva fronteira de Pareto. Por fim, é necessário encontrar, de entre todos os pontos presentes na curva, o que melhor se ajusta a todos os objetivos em estudo.

No modelo MF em análise existem dois objetivos concorrentes, a percentagem de FIFO cumprida e o tempo total de extração. Tal como Marler & Arora (2004) afirmaram, não é possível encontrar uma solução ótima que melhore simultaneamente o valor destes dois objetivos. Assim, foi necessário, numa primeira fase, encontrar o conjunto de pontos pertencentes à curva fronteira de Pareto para o modelo em estudo. Para este efeito, utilizou-se o método da função objetivo limitada no qual apenas um dos objetivos está presente na função objetivo, sendo os restantes adaptados para restrições adicionais ao modelo (Marler & Arora, 2004).

Assim, com o intuito de determinar a curva fronteira de Pareto, selecionou-se a configuração 1, por representar um layout possível do Shopstocker, e aleatoriamente foi selecionada uma sequência de extração das 30 anteriormente estudadas. De seguida, com os dados devidamente importados para o

ambiente de desenvolvimento integrado PyCharm, determinaram-se os extremos, isto é, os valores ótimos para cada objetivo individualmente, sem a presença do outro objetivo. O valor máximo de FIFO que é possível cumprir na instância utilizada é de 98% (2% de complementar de FIFO), implicando um tempo de extração de 9003 segundos. Por outro lado, o valor mínimo do tempo de extração é 7565 segundos, sendo obtido com um FIFO de 83% (17% complementar de FIFO).

Após a determinação dos valores extremos, utilizou-se o método da função objetivo limitada para se obter o conjunto de pontos pertencentes à curva fronteira de Pareto. Na função objetivo minimizou-se o tempo nominal de extração, ficando limitada por uma restrição a percentagem de FIFO mínima a cumprir. Realizaram-se catorze experiências entre os dois pontos extremos, iniciando com a percentagem de FIFO mínima de 83%, até ao valor 98%.

Com os valores registados do tempo de extração e do complementar da percentagem de FIFO cumprida em cada experiência (100 – percentagem de FIFO), construiu-se o gráfico exposto na Figura 26 que representa a curva fronteira de Pareto para a instância desenvolvida.

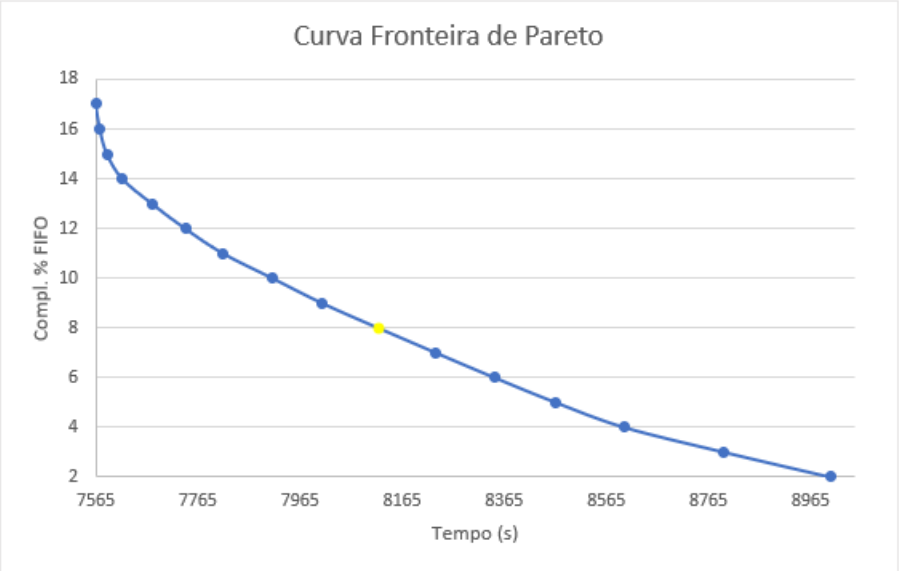

Figura 26 - Curva fronteira de Pareto.

Com a análise dos valores obtidos, assim como do gráfico desenvolvido, procurou-se realizar um balanço entre os dois KPI's, isto é, procurou-se encontrar o melhor ponto que maximizava a percentagem de FIFO cumprida, sem comprometer o tempo total de extração e vice-versa. Para este efeito, analisaram-se os diversos pontos da curva fronteira de Pareto, no sentido de determinar o ponto correspondente ao valor mínimo da raiz quadrada da soma do tempo de extração ao quadrado com o

complementar da percentagem de FIFO ao quadrado. Assim, com o estudo realizado, determinou-se o ponto assinalado a amarelo na Figura 26, correspondente a uma percentagem de 8% do complementar do FIFO e um tempo total de extração de 8118 segundos, como sendo o que melhor se ajustava a esta instância.

Por fim, para se determinar as ponderações que melhor se ajustavam aos dois objetivos em estudo realizaram-se várias experiências, até se atingirem, no *output* do modelo, os valores do tempo total de extração e da percentagem de FIFO cumprida, correspondentes ao melhor ponto anteriormente determinado. Para que o somatório dos dois pesos fosse igual a um, e analisando as unidades em que cada objetivo se encontrava, definiram-se as ponderações de 0,9912 para a percentagem de FIFO a cumprir e de 0,0088 para o tempo de extração. A discrepância entre estes dois pesos deve-se à diferente ordem de grandeza das escalas da percentagem de FIFO em percentagem, [83 a 98], e do tempo de extração em segundos, [7565 a 8965].

5.4 Modelo MTFM – Efeito da Customização

A crescente pressão do mercado obriga a que as empresas lidem com produtos que respondam de forma personalizada às pretensões dos seus clientes. Por este motivo, a empresa do setor automóvel em estudo oferece vários modelos de PC com diferentes pormenorizações. De acordo com a análise ABC efetuada, das mais de cem referências habitualmente em armazenamento, apenas vinte e duas têm elevada procura e, cerca de trinta e cinco procura intermédia. Significa isto que, existem regularmente em armazenamento quase cinquenta referências com baixa procura. Uma vez que a planta estudada do Shopstocker já se encontra em funcionamento, o seu design e estrutura não podem ser alterados sendo por isso necessário distribuir mais de cem referências em 86 linhas de armazenamento. A esta dificuldade acresce o facto de o piso inferior estar exclusivamente reservado a PC da classe *high runners*. Assim, é inevitável que os PC das classes *medium* e *low runners* estejam distribuídos de forma mais ou menos caótica pelas linhas do piso superior.

Por todas estas razões, o processo de extração de PC *high runners* é imediato uma vez que, o PC mais próximo da saída é também o primeiro da ordem FIFO, e não obriga à movimentação extra de outros PC por impedimento físico. Segundo a análise efetuada aos dados do período analisado, os *high runners* correspondem a 78,5% do total de extrações. Este valor é consentâneo com a percentagem de FIFO cumprida pelo sistema real de 75%. Por outro lado, como dentro das classes *medium* e *low runners* o armazenamento não é exclusivo, a extração de um determinado PC, obriga à perda, para a encomenda

em curso, de alguns PC por impedimento físico. Por esta razão, o processo de extração de PC destas classes é mais complexo e tem um impacto negativo no número de PC movimentados e no cumprimento do FIFO. Os PC das classes *medium* e *low runners* correspondem, respetivamente, a 14,1% e 7,4% do total de extrações.

De forma a estudar o efeito enunciado acima, realizaram-se dois conjuntos de experiências que serão descritas de seguida. Nesta fase do projeto pretendeu-se avaliar o efeito da customização na eficiência do Shopstocker, nomeadamente no tempo total de extração, na percentagem da regra FIFO cumprida e no número de PC movimentados, sem, no entanto, procurar quantificar os valores de forma mais correta do ponto de vista estatístico.

Os dois conjuntos de experiências serviram para identificar e confirmar se os mesmos padrões de comportamento das medidas de desempenho estudadas se verificavam em ambos. Em cada experiência foram realizadas as seguintes etapas:

1. Definição de uma sequência de referência com 100 PC da classe *high runner*.
2. Submissão da sequência definida no ponto anterior à configuração 1 e recolha dos valores dos KPI's.
3. Inserção sucessiva de PC *medium runner* à sequência definida no ponto 1, até se atingirem as percentagens de *medium runners* existentes nas sequências reais.
4. Submissão das sequências definidas no ponto 3 à configuração 1 e recolha dos valores dos KPI's.
5. Inserção sucessiva de PC *low runner* à sequência definida no ponto 1, até se atingirem as percentagens de *low runners* existentes nas sequências reais.
6. Submissão das sequências definidas no ponto 5 à configuração 1 e recolha dos valores dos KPI's.
7. Submissão de sequências representativas das percentagens reais de cada classe e avaliação dos KPI's.

Nestas experiências a função objetivo utilizada no modelo MTFM foi definida de forma a minimizar o valor das três medidas de desempenho estudadas: tempo total de extração, complementar da percentagem da regra FIFO cumprida e número total de PC movimentados.

Com este conjunto de etapas realizaram-se as experiências pretendidas e construiu-se a Tabela 10 e a Tabela 11 que identificam em cada linha a experiência realizada.

Tabela 10 - Resultados das experiências 1 relativas ao efeito da customização.

Sequência						KPI									
nº High	%	nº Mid	%	nº Low	%	tempo	%	acumulado	FIFO	%	acumulado	mov	%	acumulado	
100	100,0%	0	0,0%	0	0,0%	6550	0,0%	0,0%	100	100,0%	0,0%	0	0,0%	0,0%	
99	99,0%	1	1,0%	0	0,0%	6644	1,4%	1,4%	100	0,0%	0,0%	3	3,0%	3,0%	Inserir Medium Runners
98	98,0%	2	2,0%	0	0,0%	6765	1,8%	3,2%	100	0,0%	0,0%	5	1,9%	4,9%	
97	97,0%	3	3,0%	0	0,0%	6755	-0,1%	3,1%	99	1,0%	1,0%	5	0,0%	4,9%	
96	96,0%	4	4,0%	0	0,0%	6833	1,1%	4,2%	99	0,0%	1,0%	6	1,0%	5,9%	
95	95,0%	5	5,0%	0	0,0%	6929	1,4%	5,6%	99	0,0%	1,0%	8	1,9%	7,8%	
94	94,0%	6	6,0%	0	0,0%	7004	1,1%	6,7%	99	0,0%	1,0%	8	0,0%	7,8%	
93	93,0%	7	7,0%	0	0,0%	6997	-0,1%	6,6%	98	1,0%	2,0%	9	0,9%	8,7%	
92	92,0%	8	8,0%	0	0,0%	7090	1,3%	7,9%	98	0,0%	2,0%	10	0,9%	9,6%	
91	91,0%	9	9,0%	0	0,0%	7206	1,6%	9,5%	98	0,0%	2,0%	12	1,8%	11,4%	
90	90,0%	10	10,0%	0	0,0%	7253	0,6%	10,1%	97	1,0%	3,0%	21	8,0%	19,5%	
89	89,0%	11	11,0%	0	0,0%	7220	-0,5%	9,7%	95	2,1%	5,1%	20	-0,8%	18,7%	
88	88,0%	12	12,0%	0	0,0%	7010	-3,0%	6,7%	92	3,2%	8,3%	20	0,0%	18,7%	
87	87,0%	13	13,0%	0	0,0%	7107	1,4%	8,0%	92	0,0%	8,3%	19	-0,8%	17,8%	
86	86,0%	14	14,0%	0	0,0%	7192	1,2%	9,2%	92	0,0%	8,3%	19	0,0%	17,8%	
85	85,0%	15	15,0%	0	0,0%	7159	-0,5%	8,8%	91	1,1%	9,3%	18	-0,8%	17,0%	
99	99,0%	0	0,0%	1	1,0%	6494	-0,9%	-0,9%	100	0,0%	0,0%	3	3,0%	3,0%	Inserir Low Runners
98	98,0%	0	0,0%	2	2,0%	6449	-0,7%	-1,6%	99	1,0%	1,0%	3	0,0%	3,0%	
97	97,0%	0	0,0%	3	3,0%	6435	-0,2%	-1,8%	98	1,0%	2,0%	10	6,8%	9,8%	
96	96,0%	0	0,0%	4	4,0%	6489	0,8%	-0,9%	97	1,0%	3,0%	23	11,8%	21,6%	
95	95,0%	0	0,0%	5	5,0%	6453	-0,6%	-1,5%	98	-1,0%	2,0%	31	6,5%	28,1%	
94	94,0%	0	0,0%	6	6,0%	6370	-1,3%	-2,8%	98	0,0%	2,0%	30	-0,8%	27,4%	
93	93,0%	0	0,0%	7	7,0%	6354	-0,3%	-3,1%	96	2,0%	4,0%	38	6,2%	33,5%	
92	92,0%	0	0,0%	8	8,0%	6338	-0,3%	-3,3%	95	1,0%	5,1%	41	2,2%	35,7%	
78	78,0%	14	14,0%	8	8,0%	6980		6,2%	87		13,0%	60		37,5%	Seq Real
78	78,0%	15	15,0%	7	7,0%	7044		7,0%	87		13,0%	58		36,7%	
79	79,0%	14	14,0%	7	7,0%	7077		7,4%	88		12,0%	59		37,1%	

A primeira experiência, contendo 100 PC da classe *high runner*, foi utilizada como referência. As primeiras seis colunas identificam o número de PC por classe e a sua percentagem em função do total. As restantes colunas registam os valores dos três KPI's registados em cada experiência de forma individualizada para os seguintes conjuntos: inserção de *medium runners*, inserção de *low runners* e, por fim, submissão de sequências aproximadas às percentagens reais de cada classe. O efeito nas medidas de desempenho que a inserção de cada *medium* ou *low runner* teve, em percentagem, relativamente à experiência imediatamente anterior está registado nas colunas identificadas como *%* enquanto que o efeito em comparação com a experiência de referência realizada está exposto nas colunas identificadas como *acumulado*.

O modelo MTFM apresentou tempos médios de CPU de 45 segundos para resolver o problema e as instâncias submetidas nestas experiências.

De acordo com os resultados obtidos, em ambas as sequências de referência, apenas contendo PC *high runners*, foi possível cumprir 100% do FIFO e não se movimentaram PC extra por impedimento físico

para além dos 100 da sequência de extração. Por outro lado, verifica-se, nas duas instâncias concebidas, a existência de um padrão semelhante de diminuição da percentagem de FIFO cumprida e de aumento do número de PC movimentados, à medida que são adicionados *medium* e *low runners* à sequência de extração. Para além dos padrões de evolução identificados acima, foi possível ainda observar que, a inserção de PC da classe *low runner*, obriga à movimentação de mais PC quando comparada com a adição de PC *medium runners*. Por outro lado, a percentagem de perda de FIFO com a extração de PC destas duas classes é semelhante.

Tabela 11 - Resultados das experiências 2 relativas ao efeito da customização.

Sequência						KPI									
nº High	%	nº Mid	%	nº Low	%	tempo	%	acumulado	FIFO	%	acumulado	mov	%	acumulado	
100	100,0%	0	0,0%	0	0,0%	7550	0,0%	0,0%	100	100,0%	0,0%	0	0,0%	0,0%	
99	99,0%	1	1,0%	0	0,0%	7639	1,2%	1,2%	100	0,0%	0,0%	3	3,0%	3,0%	Inserir Medium Runners
98	98,0%	2	2,0%	0	0,0%	7717	1,0%	2,2%	100	0,0%	0,0%	4	1,0%	4,0%	
97	97,0%	3	3,0%	0	0,0%	7700	-0,2%	2,0%	99	1,0%	1,0%	5	1,0%	4,9%	
96	96,0%	4	4,0%	0	0,0%	7760	0,8%	2,7%	99	0,0%	1,0%	5	0,0%	4,9%	
95	95,0%	5	5,0%	0	0,0%	7714	-0,6%	2,1%	97	2,0%	3,0%	13	7,6%	12,6%	
94	94,0%	6	6,0%	0	0,0%	7804	1,2%	3,3%	97	0,0%	3,0%	13	0,0%	12,6%	
93	93,0%	7	7,0%	0	0,0%	7546	-3,4%	-0,1%	94	3,1%	6,1%	12	-0,9%	11,7%	
92	92,0%	8	8,0%	0	0,0%	7518	-0,4%	-0,5%	93	1,1%	7,2%	11	-0,9%	10,8%	
91	91,0%	9	9,0%	0	0,0%	7490	-0,4%	-0,9%	92	1,1%	8,3%	10	-0,9%	9,9%	
90	90,0%	10	10,0%	0	0,0%	7452	-0,5%	-1,4%	91	1,1%	9,3%	9	-0,9%	9,0%	
89	89,0%	11	11,0%	0	0,0%	7535	1,1%	-0,3%	91	0,0%	9,3%	10	0,9%	9,9%	
88	88,0%	12	12,0%	0	0,0%	7602	0,9%	0,6%	91	0,0%	9,3%	9	-0,9%	9,0%	
87	87,0%	13	13,0%	0	0,0%	7680	1,0%	1,6%	91	0,0%	9,3%	10	0,9%	9,9%	
86	86,0%	14	14,0%	0	0,0%	7655	-0,3%	1,3%	90	1,1%	10,4%	10	0,0%	9,9%	
85	85,0%	15	15,0%	0	0,0%	7761	1,4%	2,6%	90	0,0%	10,4%	12	1,8%	11,7%	
99	99,0%	0	0,0%	1	1,0%	7577	0,4%	0,4%	99	1,0%	1,0%	9	9,0%	9,0%	Inserir Low Runners
98	98,0%	0	0,0%	2	2,0%	7504	-1,0%	-0,6%	98	1,0%	2,0%	13	3,7%	12,7%	
97	97,0%	0	0,0%	3	3,0%	7417	-1,2%	-1,8%	98	0,0%	2,0%	14	0,9%	13,6%	
96	96,0%	0	0,0%	4	4,0%	7327	-1,2%	-3,0%	98	0,0%	2,0%	14	0,0%	13,6%	
95	95,0%	0	0,0%	5	5,0%	7244	-1,1%	-4,2%	97	1,0%	3,0%	13	-0,9%	12,7%	
94	94,0%	0	0,0%	6	6,0%	7187	-0,8%	-5,0%	96	1,0%	4,1%	14	0,9%	13,6%	
93	93,0%	0	0,0%	7	7,0%	7090	-1,4%	-6,3%	97	-1,0%	3,0%	15	0,9%	14,4%	
92	92,0%	0	0,0%	8	8,0%	7122	0,4%	-5,9%	96	1,0%	4,1%	19	3,5%	17,9%	
78	78,0%	14	14,0%	8	8,0%	7227		-4,5%	86		14,0%	29		22,5%	Seq Real
78	78,0%	15	15,0%	7	7,0%	7355		-2,7%	87		13,0%	30		23,1%	
79	79,0%	14	14,0%	7	7,0%	7249		-4,2%	87		13,0%	28		21,9%	

Relativamente ao tempo de extração dos PC requisitados foi possível perceber que, a extração de PC *medium runners* aumentou, ainda que ligeiramente, o valor desta medida de desempenho. Em contrapartida, a extração de PC *low runners* diminui o tempo total de extração.

Do mesmo modo, as experiências realizadas com as percentagens de PC de cada classe similares à situação real revelaram igualmente que existe uma quebra acentuada na percentagem de FIFO cumprida com a existência de PC das classes *medium* e *low runners* na sequência de extração. Da mesma forma, o número de PC movimentados aumenta consideravelmente em comparação à sequência de referência contendo apenas *high runners*. Por outro lado, as experiências realizadas não evidenciaram

um padrão comum às duas instâncias concebidas, relativamente ao tempo total de extração. Por esta razão, não foi possível estabelecer uma relação entre a inserção de PC das classes *medium* e *low runners* e esta medida de desempenho.

O estudo realizado ao efeito da customização permitiu concluir que se as referências pedidas na sequência de extração pertencessem exclusivamente à classe dos *high runners*, seria possível cumprir a 100% da regra FIFO e não se movimentariam quaisquer PC a mais para além dos requeridos na sequência de extração. Estes resultados vão de encontro ao afirmado por Yu & de Koster (2009), e evidenciam que as condições ideais de funcionamento do Shopstocker exigem o armazenamento dedicado, de uma referência por linha, ordenados pela regra FIFO. Desta forma, o melhor PC para abandonar o sistema seria sempre o mais próximo da saída, reduzindo o tempo de extração, o primeiro da ordem FIFO, aumentando a percentagem de FIFO cumprida, e não movimentaria qualquer PC extra não solicitado para a sequência.

Esta análise permitiu ainda perceber que a inserção de PC de classes *medium* e *low runners* na sequência de extração reduz significativamente a percentagem de FIFO cumprida e aumenta o número de PC movimentados. A necessidade de armazenar 102 referências em 86 linhas do Shopstocker, obriga a que várias linhas de armazenamento sejam caóticas. Esta mistura de referências torna o processo de seleção do melhor PC para abandonar o sistema mais complexo, uma vez que extrair o primeiro PC da ordem FIFO pode ter um custo elevado no tempo de extração e no número de PC movimentados, dificultando o cumprimento do *takt time*.

Assim, concluiu-se que o efeito de customização, ou seja, a necessidade de a empresa disponibilizar um elevado número de modelos de PC tem um impacto claro na perda de eficiência do Shopstocker. O aumento do tempo total de extração, a diminuição da percentagem de FIFO cumprida e o aumento do número de PC movimentados são os custos da elevada personalização dos modelos de PC.

6. CONCLUSÕES

Neste capítulo são apresentadas as principais conclusões resultantes do projeto realizado. Na fase final deste capítulo são ainda identificadas as limitações deste projeto e são apresentadas recomendações para o futuro do Shopstocker e sistemas de armazenamento semelhantes.

6.1 Considerações Finais

Apesar da tendência de redução de inventários por parte das empresas, a utilização de sistemas de armazenamento é inevitável em determinadas situações. Por esta razão, é decisivo encontrar para cada situação o armazém mais eficiente e que melhor responda às pretensões de cada organização. A necessidade de um sistema para armazenamento de para-choques (PC) levou ao desenvolvimento e implementação do Shopstocker. Este armazém automático de alta densidade permite armazenar PC finalizados ao longo das suas linhas de armazenamento até que sejam requisitados numa encomenda. Os PC são inseridos no sistema numa zona de *input* e tracionados por um circuito em carrossel até às linhas de armazenamento. Nestas linhas os PC fluem da extremidade de inserção até à extremidade de extração através da força gravítica. Quando um PC é selecionado para suprir uma determinada encomenda é extraído da linha de armazenamento e tracionado até uma zona de *output*. Este armazém compacto é ideal para o armazenamento de uma referência por linha. No entanto, devido à elevada customização existe um grande número de referências distintas disponibilizadas e que têm de ser armazenadas no Shopstocker. Uma vez que o número de linhas de armazenamento é menor que o número de referências disponibilizadas, foi necessário implementar uma política de armazenamento baseada na procura dos vários modelos. Assim, a empresa definiu três classes de PC: *high runners*, com elevada procura; *medium runners*, com procura intermédia e *low runners*, com baixa procura. Nas linhas destinadas a armazenar PC da classe dos *high runners* não existe mistura de referências pelo que, cada linha é exclusiva de uma só referência. Por outro lado, nas linhas destinadas a armazenar *medium* e *low runners* é possível misturar até 3 e 25 referências distintas, respetivamente. Quando o sistema recebe uma encomenda, necessita de selecionar, de entre todas as possibilidades, o PC que se encontra mais perto da zona de *output* e que está há mais tempo no sistema. Nas linhas dedicadas a *high runners* a seleção é imediata uma vez que o PC mais próximo da saída é também o mais antigo no sistema. No entanto, nas linhas mistas e caóticas destinadas aos *medium* e *low runners*, respetivamente, o processo de seleção torna-se muito mais complexo. Este projeto procurou por isso estudar o desempenho do

Shopstocker através do desenvolvimento de algoritmos de seleção de produtos e da análise de três medidas de desempenho: tempo total de extração, percentagem de FIFO cumprida e número de PC movimentados.

A revisão bibliográfica realizada permitiu identificar as tendências para a implementação de sistemas cada vez mais automatizados e cujo funcionamento se guia pelos princípios da Indústria 4.0. Os armazéns AS/RS, em especial os de alta densidade, encontram-se amplamente estudados na literatura. No entanto, verificou-se que as investigações se centram maioritariamente nos processos e políticas de armazenamento e menos no problema de seleção de produtos para *picking* estudado neste projeto.

Com o conhecimento do problema real existente no Shopstocker e do estado de arte da literatura optou-se pelo desenvolvimento de um problema de otimização combinatória. Numa primeira fase, analisaram-se os dados recolhidos e construíram-se as instâncias necessárias. Posteriormente, desenvolveu-se um modelo matemático de programação linear inteira mista representativo das restrições e regras de funcionamento do sistema real. Desenvolveram-se 3 modelos em linguagem de programação Python: MPA, MF e MTFM. Para além dos modelos de otimização, foram desenvolvidas em Python funções que permitem obter diretamente o valor correto para os 3 KPI's a partir da solução ótima obtida com os modelos. Adicionalmente, implementou-se um algoritmo que permite observar graficamente a distribuição dos PC selecionados para a solução ótima. Com este conjunto de tarefas foi possível realizar experiências computacionais e estudar o funcionamento do Shopstocker.

Com os modelos devidamente implementados na linguagem de programação Python e com as instâncias representativas de estados de configuração do Shopstocker e de sequências de extração realizaram-se as experiências computacionais. Após a validação dos modelos e algoritmos desenvolvidos procurou-se estudar o efeito nos KPI's de uma reorganização dos PC armazenados em linhas mistas e caóticas por um maior número de linhas. Desta forma, procurou-se avaliar se a menor mistura de referências nas linhas caóticas e mistas resultava numa melhoria nos valores das medidas de desempenho estudadas. Os resultados demonstraram que apesar de existir uma melhoria significativa da percentagem de FIFO cumprida, por se perderem menos PC da ordem FIFO, existe um agravamento do tempo de extração e do número de PC movimentados. A maior dispersão dos PC pelo armazém leva ao aumento da distância a percorrer e à perda do efeito de compensação na extração de *medium* e *low runners*. Nesta experiência comprovou-se a importância do efeito de compensação para o funcionamento deste sistema de armazenamento.

Posteriormente, realizou-se uma análise multiobjetivo ao modelo MF procurando avaliar simultaneamente dois objetivos concorrentes: tempo total de extração e percentagem de FIFO cumprida. A utilização do método da função objetivo limitada e da construção da curva fronteira de Pareto permitiu perceber as ponderações que melhor se ajustavam aos dois objetivos em estudo.

Por fim, procurou-se avaliar o impacto da customização na eficiência do Shopstocker. O aumento da pressão do mercado no sentido de exigir produtos cada vez mais personalizados obriga as empresas a disponibilizarem uma maior variedade de opções. Por esta razão, o Shopstocker necessita de armazenar mais de 100 referências em simultâneo pelas suas 86 linhas. Estas experiências procuraram, por isso, identificar padrões no comportamento das medidas de desempenho à medida que eram exigidas maiores percentagens de *medium* e *low runners* nas sequências de extração. Os resultados mostraram claramente que numa sequência contendo apenas *high runners* é possível cumprir 100% de FIFO sem movimentar qualquer PC extra para além dos requeridos na sequência. Por outro lado, a inserção de PC das classes *medium* e *low runners* na sequência de extração, armazenados em linhas mistas e caóticas, respetivamente, leva a uma perda da percentagem de FIFO cumprida e a um aumento do número de PC movimentados, o que dificulta o cumprimento do *takt time*. Estes factos comprovam que a mistura de referências em linhas de armazenamento leva a uma perda de eficiência do Shopstocker.

O projeto realizado permitiu estudar este sistema de armazenamento de acordo com as suas regras de funcionamento e concluir que, no sistema atualmente em funcionamento existe uma perda clara de eficiência pela existência de linhas mistas e caóticas. A percentagem de FIFO cumprida pelo sistema real e a entropia causada pelo elevado número de PC movimentados, que resulta num aumento do tempo de extração, dificultam o cumprimento do *takt time* definido. O Shopstocker, assim como outros sistemas de armazenamento semelhantes devem procurar armazenar os produtos em linhas exclusivas e ordenados segundo a regra FIFO. Desta forma, aumenta-se a eficiência de todo o sistema e reduzem-se os custos associados à sua utilização.

A modelação desenvolvida em Python permite ainda que o algoritmo seja configurado de forma a otimizar as medidas de desempenho da percentagem de FIFO cumprida e do número de PC movimentados ficando o tempo total de extração limitado pelo *takt time* associado à extração de todos os PC da sequência. Desta forma o algoritmo de seleção de PC concebido pode ser integrado num sistema de apoio à decisão (SAD). Sempre que a empresa recebe uma encomenda pode inserir no

sistema a configuração corrente do Shopstocker como *input* para o modelo que, por sua vez, irá fornecer uma solução otimizada para a sequência de PC solicitada. O algoritmo MTFM fornece uma resposta ótima em poucos segundos quando submetida uma sequência de extração até 50 PC. Este facto é ainda mais importante sabendo que representa um aumento considerável em relação ao sistema real em funcionamento que apenas seleciona conjuntos de 8 PC para cada sequência.

6.2 Sugestões para o Shopstocker e Sistemas de Armazenamento Semelhantes

De acordo com os resultados obtidos o sistema em funcionamento pode melhorar a sua eficiência se a variedade de referências disponibilizadas for menor. Um armazenamento exclusivo permitiria aumentar a percentagem de FIFO cumprida, reduzir o número de PC movimentados e, consequentemente, diminuir o tempo total de extração.

Uma outra sugestão é a recirculação dos PC extra removidos para a extração dos requeridos na sequência, que possibilitaria que estes PC fizessem parte das opções de seleção para a encomenda em preparação. Com esta estratégia os PC corretos da ordem FIFO ficariam disponíveis para que o sistema os escolhesse e a percentagem de cumprimento desta medida de desempenho iria aumentar.

De modo a não agravar o funcionamento do sistema existente recomenda-se ainda que durante os períodos de inatividade os PC possam ser reorganizados para que a configuração mais próxima do ideal seja mantida constante.

De acordo com o estudo realizado as linhas do piso superior mais próximas da zona de *output* aparentam ter um tempo de extração menor do que as linhas do piso inferior mais afastadas da zona de *output*. Por esta razão, poderá ser vantajoso estudar a possibilidade de armazenar PC da classe *high runner* no piso superior uma vez que o tempo total de extração poderá sofrer uma redução.

Para sistemas de armazenamento futuros semelhantes ao estudado neste projeto recomenda-se que existam linhas de armazenamento exclusivas evitando as perdas de eficiência identificadas neste projeto. A fase de *design* do armazém deverá ter em consideração a quantidade de referências distintas que serão armazenadas.

Em situações em que a mistura de referências seja inevitável, recomenda-se a construção de um sistema de armazenamento com linhas de tamanho variável. Desta forma, seria possível ter linhas de maior dimensão exclusivas de uma só referência e linhas de tamanho mais reduzido com mistura de referências. Assim, evitar-se-iam situações de excesso de mistura de referências e facilitar-se-ia o acesso aos PC armazenados em linhas caóticas.

6.3 Limitações do Projeto e Trabalho Futuro

Na realização deste projeto existiram algumas limitações associadas à obtenção de dados do sistema real. As configurações desenvolvidas neste projeto foram geradas de forma indireta a partir dos dados existentes, uma vez que não se conseguiu obter uma configuração real do estado do Shopstocker.

Uma outra limitação consistiu no facto de se considerar o Shopstocker numa configuração estática inicial, não sendo possível ajustar a configuração a uma sequência específica. O sistema de armazenamento real é dinâmico e a cada instante são inseridos, realocados e extraídos PC no sistema.

O sistema de armazenamento estudado neste projeto de dissertação apresenta uma gestão operacional muito complexa e dinâmica. A organização dos PC no interior do Shopstocker está dependente de diversos fatores tais como: PC que entram no sistema, PC que saem do sistema, realocação dos PC e suportes vazios, entre outros. Por esta razão, um estudo através da simulação discreta permitiria realizar uma avaliação exaustiva da dinâmica do Shopstocker e das suas regras funcionais e permitiria perceber com maior detalhe a evolução dos valores das medidas de desempenho ao longo do tempo.

REFERÊNCIAS BIBLIOGRÁFICAS

Barreto, L., Amaral, A., & Pereira, T. (2017). Industry 4.0 implications in logistics: an overview. *Procedia Manufacturing*, *13*, 1245–1252. https://doi.org/10.1016/j.promfg.2017.09.045

Bengü, G. (1995). An optimal storage assignment for automated rotating carousels. *IIE Transactions (Institute of Industrial Engineers)*, *27*(1), 105–107. https://doi.org/10.1080/07408179508936722

Berry, J. R. (1968). Elements of warehouse layout. *International Journal of Production Research*, *7*(2), 105–121. https://doi.org/10.1080/00207546808929801

Brasil, P. (2019). Pinto Brasil. Retrieved January 10, 2020, from https://pintobrasil.com/

Cardin, O., Castagna, P., Sari, Z., & Meghelli, N. (2012). Performance evaluation of In-Deep Class Storage for Flow-Rack AS/RS. *International Journal of Production Research*, *50*(23), 6775–6791. https://doi.org/10.1080/00207543.2011.624561

Clark, A. W., & Trist, E. L. (1976). Action Research and Adaptive Planning. In *Experimenting with Organizational Life*. https://doi.org/10.1007/978-1-4613-4262-5_17

Cormier, G., & Gunn, E. A. (1992). A review of warehouse models. *European Journal of Operational Research*. https://doi.org/10.1016/0377-2217(92)90231-W

de Koster, R., Le-Duc, T., & Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, *182*(2), 481–501. https://doi.org/10.1016/j.ejor.2006.07.009

Dolgui, A., & Proth, J. M. (2010). Supply chain engineering: Useful methods and techniques. In *Supply Chain Engineering: Useful Methods and Techniques*. https://doi.org/10.1007/978-1-84996-017-5

Fiel, J. (2010). Bússola. Retrieved January 10, 2020, from https://bussola.blogs.sapo.pt/150735.html

Fusko, M., Rakyta, M., & Manlig, F. (2017). Reducing of Intralogistics Costs of Spare Parts and Material of Implementation Digitization in Maintenance. *Procedia Engineering*, *192*, 213–218. https://doi.org/10.1016/j.proeng.2017.06.037

Gharehgozli, A. H., Yu, Y., Zhang, X., & De Koster, R. (2014). Polynomial time algorithms to minimize total travel time in a two-depot automated storage/retrieval system. *Transportation Science*, *51*(1), 19–33. https://doi.org/10.1287/trsc.2014.0562

Ghomri, L., & Sari, Z. (2015). Mathematical modeling of retrieval travel time for flow-rack automated storage and retrieval systems. *IFAC-PapersOnLine*, *28*(3), 1906–1911. https://doi.org/10.1016/j.ifacol.2015.06.365

Goetschalckx, M., & Ratliff, D. H. (1990). Shared storage policies based on the duration stay of unit loads. *Management Science*, *36*(9), 1120–1132. https://doi.org/10.1287/mnsc.36.9.1120

Gong, Y., Zhang, Z., & Wang, S. (2009). *Stochastic Modelling and Analysis of Warehouse Operations*.

Gu, J., Goetschalckx, M., & McGinnis, L. F. (2010). Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operational Research*, *203*(3), 539–549. https://doi.org/10.1016/j.ejor.2009.07.031

Gue, K. R. (2006). Very high density storage systems. *IIE Transactions (Institute of Industrial Engineers)*, *38*(1), 79–90. https://doi.org/10.1080/07408170500247352

Gue, K. R., & Kim, B. S. (2007). Puzzle-based storage systems. *Naval Research Logistics*. https://doi.org/10.1002/nav.20230

Gupta, J. N. D. (1986). Flowshop Schedules With Sequence Dependent Setup Times. *Journal of the Operations Research Society of Japan*, *29*(3), 206–219. https://doi.org/10.15807/jorsj.29.206

Halim, N. H. A., Jaffar, A., Yusoff, N., & Adnan, A. N. (2012). Gravity Flow Rack's material handling system for Just-in-Time (JIT) production. *Procedia Engineering*, *41*(Iris), 1714–1720. https://doi.org/10.1016/j.proeng.2012.07.373

Handfield, R. B., & Nichols Jr., H. L. (1999). *Introduction to Supply Chain Management* (Prentice-Hall, Ed.). New Jersey.

Hart, W. E., Laird, C., Watson, J.-P., & Woodruff, D. L. (2017). *Pyomo – Optimization Modeling in Python (Second Edition)* (Vol. 67). https://doi.org/10.1007/978-1-4614-3226-5

Helper, S., Martins, R., & Seamans, R. (2019). Who Profits from Industry 4.0? Theory and Evidence from the Automotive Industry. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3377771

Hermann, M., Pentek, T., & Otto, B. (2016). Design principles for industrie 4.0 scenarios. *Proceedings of the Annual Hawaii International Conference on System Sciences*, *2016-March*, 3928–3937. https://doi.org/10.1109/HICSS.2016.488

Hofmann, E., & Rüsch, M. (2017). Industry 4.0 and the current status as well as future prospects on logistics. *Computers in Industry*, *89*, 23–34. https://doi.org/10.1016/j.compind.2017.04.002

Hunter, J., Dale, D., Firing, E., Droettboom, M., & Team, M. development. (2012). Matplotlib. Retrieved May 11, 2020, from https://matplotlib.org/

Hur, S., Lee, Y. H., Lim, S. Y., & Lee, M. H. (2004). A performance estimation model for AS/RS by M/G/1 queuing system. *Computers and Industrial Engineering*, *46*(2), 233–241. https://doi.org/10.1016/j.cie.2003.12.007

Hwang, H., & Ha, J.-W. (1994). Pergamon An Optimal Boundary for Two Class-based Storage Assignment Policy in Carousel System 2 . Basis of the cycle time model 3 . Development of the cycle time models for case I. *Engineering*, *27*(94), 2–5.

IBM. (2020). docplex-doc. Retrieved April 20, 2020, from https://github.com/IBMDecisionOptimization/docplex-doc

JetBrains. (2020). PyCharm. Retrieved April 6, 2020, from https://www.jetbrains.com/pycharm/

Lasi, H., Fettke, P., Kemper, H. G., Feld, T., & Hoffmann, M. (2014). Industry 4.0. *Business and Information Systems Engineering*, *6*(4), 239–242. https://doi.org/10.1007/s12599-014-0334-4

Lehnfeld, J., & Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research*, *239*(2), 297–312. https://doi.org/10.1016/j.ejor.2014.03.011

Lewin, K. (1946). Action Research and Minority Problems. *Journal of Social Issues*, (2), 34–46. Retrieved from http://bscw.wineme.fb5.uni-siegen.de/pub/nj_bscw.cgi/d759359/5_1_ActionResearchandMinortyProblems.pdf

Litvak, N., & Vlasiou, M. (2010). A survey on performance analysis of warehouse carousel systems. *Statistica Neerlandica*, *64*(4), 401–447. https://doi.org/10.1111/j.1467-9574.2010.00454.x

Luthra, S., & Mangla, S. K. (2018). Evaluating challenges to Industry 4.0 initiatives for supply chain sustainability in emerging economies. *Process Safety and Environmental Protection*, *117*, 168–179. https://doi.org/10.1016/j.psep.2018.04.018

Lutz, M. (2009). Learning Python: Powerful Object-Oriented Programming. In *Book*.

Manzini, R. (2012). Warehousing in the global supply chain: Advanced models, tools and applications for storage systems. In *Warehousing in the Global Supply Chain: Advanced Models, Tools and Applications for Storage Systems* (Vol. 9781447122). https://doi.org/10.1007/978-1-4471-2274-6

Marler, R. T., & Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, *26*(6), 369–395. https://doi.org/10.1007/s00158-003-0368-6

Mentzer, J., Dewitt, W., Keebler, J., Min, S., Nix, N., Smith, C., & Zacharia, Z. (2001). Defining Supply Chain Management. *Journal of Business Logistics*, *22*. https://doi.org/10.1002/j.2158-1592.2001.tb00001.x

Metahri, D., & Hachemi, K. (2018). Retrieval–travel-time model for free-fall-flow-rack automated storage and retrieval system. *Journal of Industrial Engineering International*, *14*(4), 807–820. https://doi.org/10.1007/s40092-018-0263-9

Pfohl, H.-C., Yahsi, B., & Kuznaz, T. (2015). The impact of Industry 4.0 on the Supply Chain. *Proceedings of the Hamburg International Conference of Logistic (HICL)-20*, (August), 32–58. https://doi.org/10.13140/RG.2.1.4906.2484

Pulp Documentation Team. (2009). Pulp. Retrieved March 30, 2020, from pythonhosted.org/PuLP

Python Software Foundation. (2020a). Docplex. Retrieved March 30, 2020, from https://pypi.org/project/docplex/

Python Software Foundation. (2020b). Python. Retrieved March 30, 2020, from https://www.python.org/

Raschka, S., & Mirjalili, V. (2019). Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. In *Packt Publishing Ltd*.

Roodbergen, K. J., & Vis, I. F. A. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, *194*(2), 343–362. https://doi.org/10.1016/j.ejor.2008.01.038

Ross, D. F. (2010). *Introduction to Supply Chain Management Technologies* (2nd ed.; Taylor & Francis Group, Ed.). Boca Raton.

Rouwenhorst, B., Reuter, B., Stockrahm, V., Van Houtum, G. J., Mantel, R. J., & Zijm, W. H. M. (2000). Warehouse design and control: Framework and literature review. *European Journal of Operational Research*, *122*(3), 515–533. https://doi.org/10.1016/S0377-2217(99)00020-X

Ruiz, R., Maroto, C., & Alcaraz, J. (2005). Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, *165*(1), 34–54. https://doi.org/10.1016/j.ejor.2004.01.022

Sari, Z., Grasman, S. E., & Ghouali, N. (2007). Impact of pickup/delivery stations and restoring conveyor locations on retrieval time models of flow-rack automated storage and retrieval systems. *Production Planning and Control*, *18*(2), 105–116. https://doi.org/10.1080/09537280600909494

Stringer, E. T. (2007). Theory & Principles of Action Research. In *Action Research*.

Tjahjono, B., Esplugues, C., Ares, E., & Pelaez, G. (2017). What does Industry 4.0 mean to Supply Chain? *Procedia Manufacturing*, *13*, 1175–1182. https://doi.org/10.1016/j.promfg.2017.09.191

Verebová, V. (2016). Improving of Material Flow in Automobile Enterprise. *Acta Logistica*, *3*(3), 5–8. https://doi.org/10.22306/al.v3i3.67

Vickson, R. G., & Fujimoto, A. (1996). Optimal storage locations in a carousel storage and retrieval system. *Location Science*, *4*(4), 237–245. https://doi.org/10.1016/s0966-8349(97)00003-x

Vol, E. (1994). Pergamon An Optimal Boundary for Two Class-based Storage Assignment Policy in Carousel System 2 . Basis of the cycle time model 3 . Development of the cycle time models for case I. *Engineering*, *27*(94), 2–5.

Wang, K., Yang, Y., & Li, R. (2019). Travel time models for the rack-moving mobile robot system. *International Journal of Production Research*, *0*(0), 1–19. https://doi.org/10.1080/00207543.2019.1652778

Witkowski, K. (2017). Internet of Things, Big Data, Industry 4.0 - Innovative Solutions in Logistics and Supply Chains Management. *Procedia Engineering*, *182*, 763–769. https://doi.org/10.1016/j.proeng.2017.03.197

Yu, Y., & de Koster, R. (2009). Optimal zone boundaries for two-class-based compact three-dimensional automated storage and retrieval systems. *IIE Transactions (Institute of Industrial Engineers)*, *41*(3), 194–208. https://doi.org/10.1080/07408170802375778

Yu, Y., & De Koster, R. B. M. (2012). Sequencing heuristics for storing and retrieving unit loads in 3D compact automated warehousing systems. *IIE Transactions (Institute of Industrial Engineers)*, *44*(2), 69–87. https://doi.org/10.1080/0740817X.2011.575441

Zaerpour, N., Yu, Y., & De Koster, R. B. M. (2015). Storing Fresh Produce for Fast Retrieval in an Automated Compact Cross-Dock System. *Production and Operations Management*. https://doi.org/10.1111/poms.12321

ANEXO I - IMPORTAÇÃO DOS DADOS PARA O PYCHARM

Após a instalação no computador pessoal do Python e do ambiente de desenvolvimento integrado PyCharm, e com o ficheiro de dados Excel construído, procedeu-se à criação de um novo projeto Python. Esta funcionalidade do PyCharm permite de forma rápida e eficiente gerar um conjunto de pastas subdivididas em diferentes secções. Dentro do projeto criado, "*projetodissertacao*", é possível gerar novas *scripts* Python, construir novos módulos ou instalar bibliotecas já existentes. Assim, e para facilitar o acesso aos dados, foi necessário gravar todos os ficheiros Excel anteriormente descritos, na pasta referente ao projeto, "*projetodissertacao*". De seguida, dentro do mesmo projeto, criou-se um ficheiro Python, "*pulp_model.py*", para o qual se pôde importar os dados e construir o modelo de otimização combinatória com a biblioteca Pulp. A Figura 27 ilustra o projeto gerado assim como os seus ficheiros.

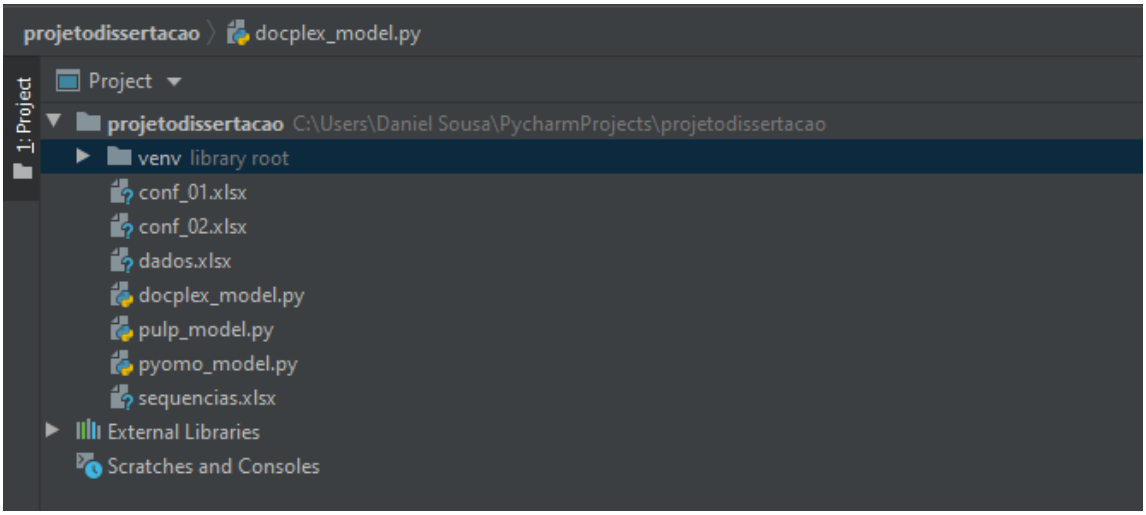

Figura 27 - Projeto "projetodissertacao" criado no PyCharm.

Após a criação do ficheiro "*pulp_model.py*", procedeu-se à importação dos dados para o ambiente de desenvolvimento integrado PyCharm. Este procedimento de importação a partir de uma fonte externa, permite que os dados sejam declarados e utilizados de forma mais eficiente, durante a programação, em comparação com a alternativa de inserção manual de cada estrutura de dados no próprio ambiente de desenvolvimento. Desta forma, e com o intuito de simplificar o processo de importação dos dados, instalou-se a biblioteca Pandas através da linha de comandos do Windows e, em particular, do comando *pip install Pandas*. Esta biblioteca foi desenvolvida, e disponibilizada de forma gratuita, para facilitar a manipulação e análise de dados em Python. Com a biblioteca disponível no computador, foi necessário incorporá-la no "*projetodissertacao*" para que a sua utilização fosse possível. Para esse efeito, no ficheiro

Python criado, utilizou-se a função *import* seguida do nome da biblioteca *pandas*, como se comprova na segunda linha da Figura 28. O PyCharm possui uma funcionalidade que deteta que um módulo ainda não está instalado no projeto em curso e sugere a sua instalação. Desta forma, o ambiente de desenvolvimento integrado instalou a biblioteca necessária, previamente existente no computador pessoal, e permitiu a sua utilização. Para facilitar a escrita do código Python definiu-se o módulo *pandas* como "*pd*", através da função *as* do próprio Python.

Da mesma forma, e tal como a Figura 28 ilustra, definiu-se o ficheiro Excel "*dados.xlsx*" com o nome, *dados*, de forma a facilitar a sua identificação ao longo do código. Por simplificação, optou-se também por definir diretamente no Python alguns parâmetros, sendo eles: o total de para-choques (*N_pc*), o total de posições na sequência de saída (*N_seq*), o total de linhas do armazém (*N_lin*), o total de posições em cada linha do armazém (*N_pos*) e o total de referências (*N_ref*) O código Python alusivo a estas definições encontra-se exposto entre as linhas 4 e 15 da Figura 28.

Do mesmo modo, o conjunto de *sets* necessários à iteração nos modelos de otimização combinatória foram definidos no próprio ficheiro Python, com auxílio da função *range(A, B, C)*. Esta função permite construir um *set* de números inteiros, iniciando no primeiro elemento, A, e terminado no elemento, B - 1, com o intervalo C entre os elementos, A e B. Como demonstram as linhas 17 a 26 da Figura 28, para se gerar um número para cada componente pretendido, definiu-se o elemento A como 1 e o elemento B como sendo o respetivo parâmetro definido anteriormente mais uma unidade. A omissão do parâmetro C indica que o intervalo entre A e B é de uma unidade.

pulp_model.py

```
1
2   import pandas as pd
3
4   dados = "dados.xlsx"
5
6   # número de PC
7   N_pc = 130
8   # número de posições da sequência
9   N_seq = 10
10  # número de linhas do armazém
11  N_lin = 13
12  # número de posicoes por linha
13  N_pos = 10
14  # número de referências
15  N_ref = 15
16
17  # set de PC
18  N = range(1, N_pc + 1)
19  # set de posições na sequência
20  S = range(1, N_seq + 1)
21  # set de linhas do armazém
22  L = range(1, N_lin + 1)
23  # set de posições
24  P = range(1, N_pos + 1)
25  # set de referênncias
26  R = range(1, N_ref + 1)
27
```

Figura 28 - Parte 1 do código do documento Python "pulp_model.py".

Por fim, para concluir a fase de importação dos dados do Excel para o ambiente de desenvolvimento integrado, procedeu-se, para cada folha Excel, a um conjunto de passos: importação de um *dataframe*, definição da primeira coluna como índice de referência e, por último, conversão dos dados do *dataframe* para o formato dicionário. A Figura 29 e Figura 30 permitem observar o código construído para a importação correta de cada coluna das folhas, *armazem*, e, *refSeq*, respectivamente.

Na primeira etapa utilizou-se a função, *read_excel()*, da biblioteca pandas, em que se invocou o ficheiro Excel anteriormente declarado como, *dados*. Posteriormente, através do parâmetro, *dtype={}*, da função, *read_excel()*, identificaram-se as colunas e o tipo de dados das mesmas. A título de exemplo, na linha 29 da Figura 29, o parâmetro, "*cod": str*, indicou que os dados da coluna, "*cod*", eram do tipo *string*. O último parâmetro adicionado, *sheet_name=*, permitiu especificar a folha, do documento definido como *dados*, a partir da qual se pretendeu fazer a importação.

Após esta primeira etapa, os dados, inicialmente no ficheiro Excel, já se encontravam carregados no ambiente de desenvolvimento PyCharm. Por padrão, o *dataframe* criado gera uma coluna extra, atribuindo um novo identificador a cada registo. Uma vez que, os dados iniciais já possuiam um código

próprio, procedeu-se à segunda etapa para definir a primeira coluna do *data frame* importado como identificador único de cada registo. Na linha 30 da Figura 29, aplicou-se ao *data frame* criado na etapa anterior, *dfref*, a função, *set_index('cod', inplace=True)*, onde o primeiro parâmetro definiu a coluna, "*cod*", como identificador e o segundo, *inplace=True*, substitui a coluna criada pela coluna, "*cod*". Sem este último parâmetro, a coluna criada na importação manter-se-ia no *data frame*.

Por fim, e uma vez que as bibliotecas utilizadas neste projeto de dissertação requeriam os dados numa estrutura do tipo dicionário, realizou-se a terceira etapa. Neste passo aplicou-se a função, *to.dict()*, do *data frame* que o converte no formato de dados pretendido. A opção *['refA']*, ilustrada na linha 31 da Figura 29 permitiu ainda especificar que cada chave do dicionário teria como valor o registo correspondente na coluna, "*refA*".

Este conjunto de três etapas foi repetido sequencialmente para todos os dados existentes no ficheiro Excel. A Figura 29 ilustra a importação da folha Excel, *armazem*, pela seguinte ordem: referências dos PC em armazenamento, linhas dos PC em armazenamento, posições dos PC em armazenamento e FIFO dos PC em armazenamento.

```
27
28  # importação das referências
29  dfref = pd.read_excel(dados, dtype={'cod': str, 'refA': int}, sheet_name="armazem")
30  dfref.set_index('cod', inplace=True)
31  ref = dfref.to_dict()['refA']
32
33  # importação das linhas
34  dflin = pd.read_excel(dados, dtype={'cod': str, 'linA': int}, sheet_name="armazem")
35  dflin.set_index('cod', inplace=True)
36  lin = dflin.to_dict()['linA']
37
38  # importação das posicoes
39  dfpos = pd.read_excel(dados, dtype={'cod': str, 'posA': int}, sheet_name="armazem")
40  dfpos.set_index('cod', inplace=True)
41  pos = dfpos.to_dict()['posA']
42
43  # importação do FIFO
44  dffifo = pd.read_excel(dados, dtype={'cod': str, 'fifo': int}, sheet_name="armazem")
45  dffifo.set_index('cod', inplace=True)
46  fifo = dffifo.to_dict()['fifo']
47
```

Figura 29 - Parte 2 do código do documento Python "pulp_model.py".

Por sua vez, é possível observar na Figura 30, a importação dos restantes dados, relativos à folha, *refSeq*, pela seguinte ordem: sequência de extração das referências, posições da sequência de extração, e, por fim, o FIFO das posições da sequência de extração.

```
47
48  # importação da sequência de extração das refêrencias
49  dfseq = pd.read_excel(dados, dtype={'ord': str, 'refS': int}, sheet_name="refSeq")
50  dfseq.set_index('ord', inplace=True)
51  seq = dfseq.to_dict()['refS']
52
53  # importação das posições da sequência de extração
54  dfpossaida = pd.read_excel(dados, dtype={'id': str, 'ord': int}, sheet_name="refSeq")
55  dfpossaida.set_index('id', inplace=True)
56  psaida = dfpossaida.to_dict()['ord']
57
58  # importação da ordem de FIFO da sequência de extração
59  dfseqfifo = pd.read_excel(dados, dtype={'ord': str, 'seqfifo': int},
60                            sheet_name="refSeq")
61  dfseqfifo.set_index('ord', inplace=True)
62  seqfifo = dfpossaida.to_dict()['seqfifo']
63
```

Figura 30 - Parte 3 do código do documento Python "pulp_model.py".

Com este conjunto de processos foi possível importar os dados do ficheiro Excel e definir os restantes no próprio ambiente de desenvolvimento PyCharm. A denominação inicial do ficheiro Excel como, *dados*, permitiu que sempre que se pretendia alterar o ficheiro de dados origem, apenas fosse necessário modificar a linha de código *dados = "dados.xlsx"*, substituindo o segmento "*dados.xlsx*" pelo nome do novo documento, e adaptar os parâmetros definidos nas linhas 6 a 26 da Figura 28.

ANEXO II - MODELAÇÃO COM A BIBLIOTECA PULP

Para facilitar a construção do modelo matemático definido na secção 4.5, utilizou-se, numa fase inicial, a biblioteca Pulp. Este módulo foi escrito em linguagem Python e disponibilizado de forma gratuita para resolver problemas de otimização combinatória a partir de modelos matemáticos. A documentação do Pulp permite que a sua implementação seja fácil e acessível a programadores com e sem experiência (Pulp Documentation Team, 2009).

Após uma investigação da documentação online disponibilizada pela Pulp Documentation Team (2009), instalou-se a biblioteca no computador pessoal, com auxílio da linha de comandos do Windows e, em particular, do comando *pip install Pulp*. Tal como na biblioteca anterior, foi necessário instalar o Pulp no projeto, "*projetodissertacao*", criado no ambiente de desenvolvimento PyCharm. De seguida, inseriu-se a função, *import pulp*, no ficheiro Python anteriormente descrito, "*pulp_model.py*", como está

ilustrado na linha 64 da Figura 31. Com importação da biblioteca Pulp para o ambiente de desenvolvimento integrado foi possível invocar as funções nela disponíveis.

```
63
64  import pulp
65  import cplex as cp
66
67  # transformar set N do tipo inteiro em string
68  N = map(str, N)
69  N = list(N)
70
71  # transformar set S do tipo inteiro em string
72  S = map(str, S)
73  S = list(S)
74
75  # transformar set R do tipo inteiro em string
76  R = map(str, R)
77  R = list(R)
78
79  # transformar set L do tipo inteiro em string
80  L = map(str, L)
81  L = list(L)
82
83  # transformar set P do tipo inteiro em string
84  P = map(str, P)
85  P = list(P)
86
```

Figura 31 - Parte 4 do código do documento Python "pulp_model.py".

O estudo da documentação da biblioteca Pulp, revelou a necessidade de converter os *sets* em listas, e o seu tipo de dados de inteiro para *string*. Para esse efeito utilizaram-se duas funções, expostas na Figura 31, disponibilizadas pelo próprio Python. A função *map(A, B)* aplica a função A, a cada elemento de B. No exemplo da linha 68 da Figura 31, a cada elemento do *set* N, anteriormente definido, foi aplicada a função, *str*, do Python, que converte os dados para o tipo *string*. A segunda função aplicada, *list()*, permitiu definir os dados do *set* numa estrutura do tipo lista. É importante salientar que a linguagem Python permite estas sucessivas conversões de um determinado elemento, neste caso N, de forma direta, sem auxílio de variáveis auxiliares. Esta funcionalidade é possível porque o Python cria um objeto, e só posteriormente o categoriza com o nome definido pelo programador. Desta forma, ao executar a linha 68 da Figura 31, o Python cria primeiramente, um objeto resultante da parte direita do operador de igual, *(map(str, N))*. De seguida, executa a parte esquerda do operador, isto é, atribui ao objeto criado o nome, N. Assim, as conversões diretas das linhas 68 e 69 da Figura 31 são possíveis porque os objetos criados em ambas são distintos. O nome N atribuído, apenas serve de etiquetagem para os diferentes

objetos criados. A conversão é assim feita por criação de objetos diferentes e não por transformação do elemento N. Este processo de conversão foi repetido para todos os *sets* anteriormente definidos e expostos nas linhas 17 a 26 da Figura 28.

Concluída a fase de transformação inicial dos dados, e estando estes de acordo com as especificações do Pulp, procedeu-se à construção em linguagem Python do modelo matemático descrito na secção 4.5.

Segundo a documentação, o primeiro passo para a modelação através da biblioteca Pulp é a definição do problema. Tal como a linha 88 da Figura 32 demonstra, foi utilizada a função, *LpProblem()*, onde se atribuiu o nome, "*ProblemaAfetação*", e se especificou, com o parâmetro *LpMinimize*, que este era um problema de minimização. O problema criado ficou categorizado com a expressão *prob*.

```
86
87  # definição do problema
88  prob = pulp.LpProblem("ProblemaAfetação", pulp.LpMinimize)
89
90  # variavel binaria x
91  var_x = pulp.LpVariable.dicts('x', (N, S), cat='Binary')
92
93  # função objetivo
94  prob += pulp.lpSum([(pos[i] * lin[i]) * var_x[i][j] for i in N for j in S]),\
95          "CustoTotal"
96
97  # restrição1 cada posição j da sequência é preenchida uma e uma só vez
98  for j in S:
99      prob += pulp.lpSum([var_x[i][j] for i in N]) == 1
100
101 # retrição2 cada PC i é atribuido no máximo uma vez a uma posição j
102 for i in N:
103     prob += pulp.lpSum([var_x[i][j] for j in S]) <= 1
104
105 # restrição3 a referência do PC i alocado é igual à pretendida na posição j
106 for j in S:
107     for i in N:
108         prob += ref[i] * var_x[i][j] == seq[j] * var_x[i][j]
109
```

Figura 32 - Parte 5 do código do documento Python "pulp_model.py".

De seguida criou-se a variável binária, X*ij*, descrita na secção 4.5, que assume o valor de 1 se o PC, *i*, for alocado à posição, *j*, da sequência de extração, e 0 no caso contrário. Para esse efeito utilizou-se a função, *LpVariable.dicts()*, exposta na linha 91 da Figura 32. Neste caso particular foram acionados três parâmetros: o primeiro, "*x*", definiu o nome da variável depois de executada a otimização; o segundo,

(N, S), indicou as listas nas quais os índices, *i*, e, *j*, respetivamente, iteraram; e o último, *cat="Binary"*, especificou que a variável é do tipo binário.

Com as definições do problema e da variável binária concluídas, procedeu-se à implementação da função objetivo. Tratando-se esta do somatório de uma multiplicação entre a variável, X*ij*, e o custo da alocação de, *i*, em, *j*, optou-se por empregar a função, *lpSum()*, da biblioteca Pulp. Esta função executou o somatório do vetor nela contido. Assim, foi necessário especificar a expressão ilustrada na linha 94 da Figura 32, onde a variável, X*ij*, foi multiplicada pelo custo. Nesta fase da construção do modelo, optou-se, por simplificação, por definir o custo como a multiplicação da linha de armazenamento pela posição do PC, *i*. Dentro da função, e de acordo com a linguagem Python, foi necessário especificar as listas, N e S, onde os índices, *i*, e, *j*, iteravam respetivamente. Por fim, atribui-se o nome, "CustoTotal", à função objetivo. É importante ainda realçar que, ao problema, *prob*, definido anteriormente, adicionou-se, através do operador, +=, a função objetivo criada.

A biblioteca Pulp possibilita que as restrições sejam definidas através da função *LpConstraint()*, no entanto, por simplificação de código, aplicou-se igualmente a função *lpSum()* na implementação das duas primeiras restrições. A primeira, implementada entre as linhas 98 e 99 da Figura 32, correspondente à restrição (2) da subsecção 4.5, garantiu que cada posição da sequência, *j*, fosse ocupada obrigatoriamente uma, e uma só vez. A segunda restrição, declarada nas linhas 102 e 103 da mesma Figura, estabeleceu que cada PC, *i*, apenas pôde ser atribuído no máximo uma vez. A terceira restrição, correspondente à (4) da subsecção 4.5, foi implementada entre as linhas 106 e 108 da Figura 32, e assegurou que a referência de cada PC, *i*, guardada no dicionário, *ref*, era igual à referência pretendida na posição, *j*, da sequência de extração, registada no dicionário, *seq*. Todas estas restrições foram adicionadas ao problema através do segmento de código, *prob +=*.

Ao estudar a documentação da biblioteca Pulp percebeu-se que esta não possuía uma função para a implementação direta da expressão "*implica*" definida na restrição (5) da subsecção 4.5. Esta limitação obrigaria a que a implementação do modelo em Python fosse mais complexa e exigisse maior capacidade computacional. Por esta razão, optou-se por terminar neste ponto a implementação do problema de otimização combinatória através da biblioteca Pulp.

Para avaliar se o modelo implementado cumpria os requisitos delineados previamente, adicionaram-se as linhas de código visíveis na Figura 33. A função *solve()*, presente na linha 111 da Figura mencionada, resolve através de um algoritmo, o problema definido. Por opção escolheu-se o algoritmo CPLEX, na versão comunitária 12.10.0.1, disponibilizado gratuitamente pela IBM para a comunidade Python. Para invocar o algoritmo CPLEX, foi necessário instalar por meio da linha de comandos do

Windows a API CPLEX, através do comando, *pip install CPLEX*. Com o algoritmo disponível no computador pessoal, instalou-se igualmente no ambiente de desenvolvimento PyCharm, no projeto, "*projetodissertacao*". Tal como nas bibliotecas anteriormente descritas, foi necessário importar o CPLEX para o ficheiro Python em questão, com auxílio da função, *import cplex as cp*, visível na linha 65 da Figura 31. Desta forma, foi possível empregar a função, *Cplex().solve()*, que resolveu o problema modelado.

```
109
110  # resolve o problema com a função CPLEX
111  prob.solve(cp.Cplex().solve())
112
113  # estado da solução
114  print("Status:", pulp.LpStatus[prob.status])
115
116  # variáveis com valor diferente de 0 são exibidas
117  for v in prob.variables():
118      if v.varValue != 0:
119          print(v.name, "=", v.varValue)
120
121  # valor otimizado da função objetivo
122  print("Custo Total = ", pulp.value(prob.objective))
123
```

Figura 33 - Parte 6 do código do documento Python "pulp_model.py".

Para se verificar o estado do problema, isto é, se o solver utilizado encontrou uma solução ótima ou se, por outro lado o problema era impossível, não consolidado, não resolvido ou indefinido, foi adicionada a linha de código 114 exposta na Figura 33. Para este efeito utilizou-se a função, *LpStatus[]*, definindo como parâmetro o estado do problema, *prob.status*. Da mesma forma, para que o valor assumido por cada variável, X*ij*, fosse disponibilizado nos resultados, implementaram-se as linhas de código 117 a 119 da Figura 32. Assim, todas as variáveis criadas no problema cujo valor fosse diferente de 0, foram exibidas no *output* do programa desenvolvido. Por fim, para se observar o valor final da função objetivo, aplicou-se a função, *value()*, da biblioteca Pulp, que apresentou o valor do objeto definido. Neste caso, tal como ilustra a linha 122 da Figura 33, identificou-se a função objetivo.

Com o código na linguagem Python completamente definido, executou-se o documento "*pulp_model.py*" através do mecanismo RUN do ambiente de desenvolvimento PyCharm e obtiveram-se os resultados expostos na Figura 34. Os dados do terminal RUN do PyCharm demonstram que, para o modelo de otimização combinatória implementado, e para o ficheiro "*dados.xlsx*", foi encontrada uma solução ótima. As variáveis com valores iguais a um foram exibidas, juntamente com o valor da função

objetivo. A análise pormenorizada dos resultados permitiu ainda perceber que o modelo implementado estava correto, cumprindo todas as restrições definidas.

```
No LP presolve or aggregator reductions.
Presolve time = 0.00 sec. (0.00 ticks)
Status: Optimal
x_116_2 = 1.0
x_15_1 = 1.0
x_22_5 = 1.0
x_30_8 = 1.0
x_64_9 = 1.0
x_68_3 = 1.0
x_69_4 = 1.0
x_84_10 = 1.0
x_98_6 = 1.0
x_99_7 = 1.0
Custo Total =  74.0
```

Figura 34 - Output da execução do programa desenvolvido no documento "pulp_model.py".

ANEXO III - MODELAÇÃO COM A BIBLIOTECA PYOMO

Uma outra biblioteca utilizada para construir o problema modelado na subsecção 4.5, foi o Pyomo (*Python Optimization Modeling Objects*). Esta biblioteca é constituída por um conjunto de outros módulos para formulação em Python de problemas de otimização combinatória, passiveis de modelação matemática. O Pyomo incorpora as principais características das linguagens AML (*Algebraic Modeling Languages*) modernas, complementadas com a linguagem orientada a objetos Python (Hart et al., 2017).

Numa primeira fase, efetuou-se um estudo da documentação do Pyomo, em especial do livro de Hart et al. (2017), e procedeu-se à instalação da biblioteca. Tal como nas bibliotecas anteriormente descritas, utilizou-se primeiramente a linha de comandos do Windows e, em particular do comando *pip install pyomo*. De seguida, criou-se um novo ficheiro Python, denominado "*pyomo_model.py*", que pode ser observado na Figura 27, e instalou-se a biblioteca no ambiente de desenvolvimento PyCharm. Neste novo documento importaram-se todas as funções da biblioteca Pyomo, existentes no módulo *environ*, através do comando, *from pyomo.environ import **. Este comando, ilustrado na linha 1 da Figura 35, permitiu que todas as funções contidas no módulo invocado, ficassem disponíveis no ficheiro "*pyomo_model.py*", sem que fosse necessário o prefixo, *pyomo.environ*, antes de cada função.

A biblioteca Pyomo permite que o programador construa o modelo de forma independente dos dados, com auxílio da função, *AbstractModel()*, ou um modelo com os dados instanciados diretamente

na *script* Python, utilizando a função, *ConcreteModel()*. Neste projeto, uma vez que o código de importação dos dados já estava construído (subsecção 0), optou-se por utilizar o modelo concreto. Deste modo, as linhas 2 a 26 da Figura 35, iguais às da Figura 28, permitiram declarar no PyCharm os *sets* necessários à construção do modelo.

pyomo_model.py

```
1  from pyomo.environ import *
2  import pandas as pd
3
4  dados = "dados.xlsx"
5
6  # número de PC
7  N_pc = 130
8  # número de posições da sequência
9  N_seq = 10
10 # número de linhas do armazém
11 N_lin = 13
12 # número de posicoes por linha
13 N_pos = 10
14 # número de referências
15 N_ref = 15
16
17 # set de PC
18 N = range(1, N_pc + 1)
19 # set de posições na sequência
20 S = range(1, N_seq + 1)
21 # set de linhas do armazém
22 L = range(1, N_lin + 1)
23 # set de posições
24 P = range(1, N_pos + 1)
25 # set de referênncias
26 R = range(1, N_ref + 1)
27
```

Figura 35 - Parte 1 do código do documento Python "pyomo_model.py".

Da mesma forma, com auxílio da biblioteca Pandas, importaram-se os registos do ficheiro Excel, "*dados.xlxs*", de modo similar ao anteriormente descrito na subsecção 0. No entanto, de acordo com a documentação da biblioteca Pyomo, as chaves das estruturas de dados do tipo dicionário deveriam ser do tipo inteiro ao invés do tipo *string* da biblioteca Pulp. Por esta razão, foi necessário adaptar a função, *read_excel()*, convertendo o parâmetro, *dtype={}*, para o formato adequado.

A Figura 36 ilustra o código alusivo à importação da informação da folha, *armazem*, do documento Excel, "*dados.xlxs*", pela seguinte ordem: referências dos PC em armazenamento, linhas dos PC em armazenamento, posições dos PC em armazenamento e FIFO dos PC em armazenamento. Por sua vez, na Figura 37 são apresentadas as linhas de código referentes à importação dos dados da folha, *refSeq*,

do documento Excel, "*dados.xlxs*". A importação seguiu a seguinte ordem: sequência de extração das referências, posições da sequência de extração, e, por fim, o FIFO das posições da sequência de extração.

```
27
28  # importação das referências
29  dfref = pd.read_excel(dados, dtype={'cod': int, 'refA': int}, sheet_name="armazem")
30  dfref.set_index('cod', inplace=True)
31  ref = dfref.to_dict()['refA']
32
33  # importação das linhas
34  dflin = pd.read_excel(dados, dtype={'cod': int, 'linA': int}, sheet_name="armazem")
35  dflin.set_index('cod', inplace=True)
36  lin = dflin.to_dict()['linA']
37
38  # importação das posicoes
39  dfpos = pd.read_excel(dados, dtype={'cod': int, 'posA': int}, sheet_name="armazem")
40  dfpos.set_index('cod', inplace=True)
41  pos = dfpos.to_dict()['posA']
42
43  # importação do FIFO
44  dffifo = pd.read_excel(dados, dtype={'cod': int, 'fifo': int}, sheet_name="armazem")
45  dffifo.set_index('cod', inplace=True)
46  fifo = dffifo.to_dict()['fifo']
47
```

Figura 36 - Parte 2 do código do documento Python "pyomo_model.py".

Com os dados disponíveis no ambiente de desenvolvimento integrado PyCharm, procedeu-se à construção do modelo matemático descrito na subsecção 4.5. Em conformidade com a documentação do Pyomo, efetuou-se primeiramente a declaração do modelo através da função, *ConcreteModel()*, exposta na linha 65 da Figura 38.

Após a criação do modelo, definiu-se a variável binária, X*ij*, através da função, *Var()*, exposta na linha 68 da Figura 38, onde se especificaram os *sets* em que os índices da variável, *i*, e, *j*, iriam iterar. Adicionalmente, ativou-se o parâmetro, *within=Binary*, para definir o tipo da variável como binário. Por fim, categorizou-se a variável como, *model.x*.

Ao contrário da biblioteca Pulp, a documentação do Pyomo sugere que a implementação do modelo tire proveito das funções que o programador pode desenvolver com a linguagem Python. Por esta razão, optou-se por definir as equações do modelo em duas etapas. Na primeira, criou-se uma função com a linguagem Python que representasse a modelação matemática definida anteriormente. Na segunda fase, adicionou-se ao modelo criado e declarado como, *model*, cada uma das funções definidas na primeira etapa.

```
47
48  # importação da sequência de extração das referências
49  dfseq = pd.read_excel(dados, dtype={'ord': int, 'refS': int}, sheet_name="refSeq")
50  dfseq.set_index('ord', inplace=True)
51  seq = dfseq.to_dict()['refS']
52
53  # importação das posições da sequência de extração
54  dfpossaida = pd.read_excel(dados, dtype={'id': int, 'ord': int}, sheet_name="refSeq")
55  dfpossaida.set_index('id', inplace=True)
56  psaida = dfpossaida.to_dict()['ord']
57
58  # importação da ordem de FIFO da sequência de extração
59  dfseqfifo = pd.read_excel(dados, dtype={'ord': int, 'seqfifo': int},
60                            sheet_name="refSeq")
61  dfseqfifo.set_index('ord', inplace=True)
62  seqfifo = dfpossaida.to_dict()['seqfifo']
63
```

Figura 37 - Parte 3 do código do documento Python "pyomo_model.py".

Para implementar a função objetivo do modelo construído na subsecção 4.5, realizaram-se então duas etapas compreendidas entre as linhas 71 e 73 da Figura 38. Na primeira, definiu-se a função, *funobj*, onde se especificou que sempre que a função declarada fosse invocada, esta deveria retornar o valor da função, *sum()*, que, por sua vez, executa o somatório do vetor nela contido. Neste caso, e de acordo com a Figura 38, a função, *sum()*, executou o somatório do produto da variável, X*ij*, pelo custo de movimentação do PC, *i*, onde os índices, *i*, e, *j*, iteravam nos *sets*, N e S, respetivamente. Uma vez mais, por simplificação nesta fase de desenvolvimento, optou-se por definir o custo de movimentação como a multiplicação da linha do PC, *i*, pela posição do mesmo. Com esta função definida, criou-se o elemento, *model.OBJ*, identificado na linha 73 da Figura 38, em que se especificou o objetivo do problema através da função, *Objective()*, do Pyomo. O parâmetro, *rule=funobj*, especifica que o objetivo do modelo se encontra na função definida como, *funobj*. É importante referir que, a função, *Objective()*, do Pyomo, define por padrão a minimização do valor da função objetivo. Assim sendo, e estando esta consentânea com o que se pretendia neste projeto, não se tornou necessário ativar nenhum parâmetro extra.

Com a função objetivo definida efetuou-se a implementação das restrições existentes no modelo matemático descrito na subsecção 4.5. O modo de construção das restrições foi similar ao da função objetivo, isto é, desenrolou-se em duas fases. Numa primeira fase, definiu-se uma função que retornava a expressão adequada a cada restrição. Na fase posterior, adicionou-se ao modelo as restrições criadas através da função, *Constraint()*, da biblioteca Pyomo. O código existente entre as linhas 75 a 88 da Figura 38, ilustra a implementação das três restrições identificadas como: *model.OcuSQ*, que garantiu que cada posição, *j*, da sequência de extração fosse obrigatoriamente ocupada por um PC, *i*,

model.PChoques, que assegurou que cada PC, *i*, fosse alocado no máximo uma vez, para o conjunto de todas as posições, *j*, da sequência de extração; e por fim, *model.Ref*, que certificou que a referência do PC, *i*, correspondia à requerida na posição, *j*, na sequência de extração. Estas três restrições correspondem às (2), (3) e (4), respetivamente, definidas na subsecção 4.5.

```
63
64  # declaração do modelo
65  model = ConcreteModel()
66
67  # variável binária x
68  model.x = Var(N, S, within=Binary)
69
70  # função objetivo
71  def funobj(model):
72      return sum(model.x[i, j]*lin[i]*pos[i] for i in N for j in S)
73  model.OBJ = Objective(rule=funobj)
74
75  # restrição1 cada posição j da sequência é preenchida uma e uma só vez
76  def restricao1(model, i):
77      return sum(model.x[i, j] for j in S) <= 1
78  model.OcuSQ = Constraint(N, rule=restricao1)
79
80  # retrição2 cada PC i é atribuido no máximo uma vez a uma posição j
81  def restricao2(model, j):
82      return sum(model.x[i, j] for i in N) == 1
83  model.PChoques = Constraint(S, rule=restricao2)
84
85  # restrição3 a referência do PC i alocado é igual à pretendida na posição j
86  def restricao3(model, i, j):
87      return (ref[i] * model.x[i, j]) == (model.x[i, j] * seq[j])
88  model.Ref = Constraint(N, S, rule=restricao3)
89
```

Figura 38 - Parte 4 do código do documento Python "pyomo_model.py".

Tal como na biblioteca Pulp, a documentação do módulo Pyomo não contempla uma função para a implementação direta da expressão "*implica*" definida na restrição (5) da secção 4.5. Esta limitação obrigaria a que a implementação em Python fosse mais complexa e exigisse maior capacidade computacional para resolver o modelo. Deste modo, e de forma análoga ao realizado com a biblioteca anterior, optou-se por terminar neste ponto a implementação do problema de otimização combinatória através da biblioteca Pyomo.

Com o intuito de avaliar o modelo construído com esta biblioteca, desenvolveram-se as linhas de código expostas na Figura 39, em que se resolveu o modelo e se exibiram os resultados. Para se especificar o *solver* a utilizar, definiu-se com a função do Pyomo, *SolverFactory()*, que o algoritmo seria o CPLEX. Posteriormente, utilizou-se a função, *solve()*, visível na linha 94 da Figura 39, e ao objeto

resultante da otimização atribuiu-se o nome de resultados. Após a execução destes passos foi possível exibir o resultado da otimização com auxílio da função, *print()*, do próprio Python. Para avaliar as variáveis, a função objetivo e todas as restrições desenvolvidas, adicionou-se a função, *model.pprint()*, do Pyomo, que após a execução exibiu todo o modelo construído.

```
89
90      # resolve o problema com a função CPLEX
91      opt = SolverFactory('cplex')
92
93      # guarda e exibe os resultados da otimização
94      resultados = opt.solve(model)
95      print(resultados)
96
97      # exibe o modelo criado
98      model.pprint()
99
```

Figura 39 - Parte 5 do código do documento Python "pyomo_model.py".

Parte do *output* obtido a partir da execução do documento Python "*pyomo_model.py*" está exposto na Figura 40. Tal como a Figura ilustra, o valor 74 da função objetivo foi igual ao obtido com o modelo desenvolvido em Pulp. Estes resultados permitiram validar o modelo construído e concluir que esta implementação cumpria os requisitos previamente definidos.

```
Problem:
- Name: tmpb65cs415
  Lower bound: 74.0
  Upper bound: 74.0
  Number of objectives: 1
  Number of constraints: 1441
  Number of variables: 1301
  Number of nonzeros: 3788
  Sense: minimize
Solver:
- Status: ok
  User time: 0.02
  Termination condition: optimal
  Termination message: MIP - Integer optimal solution\x3a Objective = 7.4000000000e+01
```

Figura 40 - Output da execução do programa desenvolvido no ficheiro "pyomo_model.py".

ANEXO IV - MODELAÇÃO COM A BIBLIOTECA DOCPLEX

A terceira biblioteca utilizada para implementar o modelo matemático desenvolvido foi o Docplex, disponibilizada de acesso livre pela IBM, para a modelação e otimização em Python, através do algoritmo CPLEX. Esta biblioteca permite resolver o problema modelado num computador pessoal, com a instalação prévia da versão IBM ILOG CPLEX Optimization Studio V12.8, ou versões mais recentes, ou, em alternativa, no serviço *cloud* da própria IBM (Python Software Foundation, 2020). Neste projeto de dissertação optou-se por resolver no computador pessoal.

Numa primeira fase procedeu-se ao estudo da documentação da biblioteca Docplex disponível na plataforma GitHub (IBM, 2020). De seguida, instalou-se no computador pessoal, com auxílio da linha de comandos do Windows, e em particular do comando, *pip install docplex*, a versão Docplex 2.13.184. Com a biblioteca já disponível no computador, instalou-se no ambiente de desenvolvimento integrado PyCharm, no projeto em curso, "*projetodissertacao*". Posteriormente, criou-se um documento Python, denominado "*docplex_model.py*" que pode ser observado na Figura 27.

docplex_model.py

```
import docplex.mp.model as cpx
import pandas as pd
import matplotlib.pyplot as plt

dados = "conf_01.xlsx"
sequencias = "sequencias.xlsx"
folhaseq = "seq1"

# número de PC
N_pc = 2150
# número de posições da sequência
N_seq = 100
# número de linhas do armazém
N_lin = 86
# número de posições por linha
N_pos = 25
# número de referências
N_ref = 103

```

Figura 41 - Parte 1 do código do documento Python "docplex_model.py".

Após uma investigação inicial percebeu-se que a versão comunitária do algoritmo CPLEX disponível na biblioteca Docplex, não seria suficiente para resolver uma instância com o volume de dados existente neste projeto. Por esta razão, e de acordo com informação recolhida, instalou-se na própria biblioteca

Docplex, a versão do algoritmo CPLEX disponível no IBM ILOG CPLEX Optimization Studio V12.8. O acesso a esta versão é disponibilizado gratuitamente pela IBM para fins académicos. Este passo foi essencial para que, o algoritmo utilizado não estivesse limitado e pudesse resolver as instâncias submetidas.

Esta biblioteca permitiu implementar integralmente em linguagem Python o modelo matemático definido na subsecção 4.5. Por esta razão, as figuras ao longo desta subsecção já contemplam o código na sua versão final. Ao longo desta subsecção são identificados os três modelos desenvolvidos: MPA (Modelo Problema de Afetação), MF (Modelo FIFO) e MTFM (Modelo Tempo FIFO Movimentados). O primeiro, MPA, é similar ao problema de afetação e permitiu implementar e testar o modelo na sua fase inicial. O modelo MF incorpora a medida de desempenho relativa à percentagem de FIFO no modelo MPA. O último modelo desenvolvido, MTFM, é o mais completo e para além das restrições dos modelos MPA e MF, considera os KPI's referentes ao número de PC movimentados e ao tempo total de extração.

Tal como a Figura 41 ilustra, foi necessário invocar as bibliotecas utilizadas no desenvolvimento deste algoritmo. A biblioteca Docplex, mais concretamente o seu módulo, *mp.model*, foram definidos como, *cpx*, na primeira linha da referida Figura. Da mesma, e em consonância com o descrito na subsecção 0, utilizou-se a biblioteca Pandas para facilitar a importação dos dados referentes às duas configurações e às diferentes sequências utilizadas neste projeto. A última biblioteca Matplotlib foi utilizada na construção das figuras representativas do Shopstocker, como será explicado mais adiante.

Nas linhas 5, 6 e 7 da Figura 41, foram declarados, respetivamente, o documento Excel com os dados referentes à configuração, o ficheiro Excel com os registos associados às sequências de extração e, por fim, a folha indicativa da sequência a utilizar.

De modo similar ao descrito nos modelos definidos com as bibliotecas Pulp e Pyomo, instanciaram-se diretamente no documento Python, os parâmetros indicativos do número de: PC, posições na sequência, linhas de armazenamento, posições por linha e referências. O código presente entre as linhas 9 e 18 Figura 41, permite ilustrar esta declaração.

```
20  # set de PC
21  N = range(1, N_pc + 1)
22  # set de posições na sequência
23  S = range(1, N_seq + 1)
24  # set de linhas do armazém
25  L = range(1, N_lin + 1)
26  # set de posições
27  P = range(1, N_pos + 1)
28  # set de referênncias
29  R = range(1, N_ref + 1)
30
31  # importação das referências
32  dfref = pd.read_excel(dados, dtype={'cod': int, 'refA': int},
33                        sheet_name="armazem")
34  dfref.set_index('cod', inplace=True)
35  ref = dfref.to_dict()['refA']
36  pref = dfref['refA'].values.tolist()
37
```

Figura 42 - Parte 2 do código do documento Python "docplex_model.py".

Da mesma forma, geraram-se os *sets* necessários à iteração dos índices das listas e dos dicionários, através da função, *range()*, do próprio Python, como se pode comprovar nas linhas 20 a 29 da Figura 42.

De seguida, procedeu-se à importação dos dados, provenientes do ficheiro Excel com os registos relativos à configuração. De modo semelhante ao descrito na subsecção 0, recorrendo ao módulo Pandas e, em particular, à sua função, *read_excel()*, importaram-se os dados no formato adequado. De acordo com a documentação do Docplex, as chaves e os valores das estruturas de dados do tipo dicionário, foram definidos como sendo do tipo inteiro. As linhas 31 a 36 da Figura 42, ilustram a importação das referências existentes no Shopstocker, a partir do documento Excel, e a sua posterior conversão em estruturas do tipo dicionário e lista.

```
38  # importação das linhas
39  dflin = pd.read_excel(dados, dtype={'cod': int, 'linA': int},
40                        sheet_name="armazem")
41  dflin.set_index('cod', inplace=True)
42  lin = dflin.to_dict()['linA']
43  plin = dflin['linA'].values.tolist()
44
45  # importação das posicoes
46  dfpos = pd.read_excel(dados, dtype={'cod': int, 'posA': int},
47                        sheet_name="armazem")
48  dfpos.set_index('cod', inplace=True)
49  pos = dfpos.to_dict()['posA']
50  ppos = dfpos['posA'].values.tolist()
51
52  # importação do FIFO
53  dffifo = pd.read_excel(dados, dtype={'cod': int, 'fifo': int},
54                         sheet_name="armazem")
55  dffifo.set_index('cod', inplace=True)
56  fifo = dffifo.to_dict()['fifo']
57  pfifo = dfpos['fifo'].values.tolist()
58
```

Figura 43 - Parte 3 do código do documento Python "docplex_model.py".

A Figura 43, por sua vez, exibe o código alusivo à importação dos dados do ficheiro Excel referente à configuração do Shopstocker. A informação foi carregada para o ambiente de desenvolvimento integrado PyCharm, pela seguinte ordem: importação das linhas de armazenamento de cada PC, importação das posições de armazenamento de cada PC e, por último, importação dos valores de FIFO associados a cada PC armazenado.

Da mesma forma, importaram-se os dados relativos aos tempos de extração de cada PC armazenado. Estes carregamentos foram efetuados de forma similar ao anteriormente descrito, com a adaptação do tipo de dados de inteiro para *float*, como se comprova pela análise da Figura 44. A ordem de importação seguida na Figura citada foi a seguinte: importação dos tempos de extração totais de cada PC, importação dos tempos de extração da linha de cada PC, e importação dos tempos de extração das posições de cada PC.

```
# importação do tempo total de extração dos PC
dft_ext = pd.read_excel(dados, dtype={'cod': int, 'textracao': float},
                        sheet_name="armazem")
dft_ext.set_index('cod', inplace=True)
t_ext = dft_ext.to_dict()['textracao']
pt_ext = dft_ext['textracao'].values.tolist()

# importação do tempo de extração dos PC da linha
dftlin = pd.read_excel(dados, dtype={'cod': int, 'tLin': float},
                       sheet_name="armazem")
dftlin.set_index('cod', inplace=True)
tlin = dftlin.to_dict()['tLin']
ptlin = dflin['tLin'].values.tolist()

# importação do tempo de extração dos PC da posição
dftpos = pd.read_excel(dados, dtype={'cod': int, 'tPos': float},
                       sheet_name="armazem")
dftpos.set_index('cod', inplace=True)
tpos = dftpos.to_dict()['tPos']
ptpos = dftpos['tPos'].values.tolist()

```

Figura 44 - Parte 4 do código do documento Python "docplex_model.py".

Os dados do documento Excel relativos à configuração guardados nas folhas, *temposLin*, e, *temposPos*, anteriormente descritos, foram importados através do código exposto na Figura 45. Para cada *data frame* foi necessário adaptar a folha do documento respetiva. Ao contrário dos tempos associados aos PC anteriormente descritos, como nenhum dado temporal possuía casas decimais e por requisitos de programação, optou-se por definir estes dados como sendo do tipo inteiro. A importação visível na Figura 45 seguiu a seguinte ordem: carregamento dos dados associados a cada linha do armazém, e importação dos registos referentes a cada posição das linhas de armazenamento.

```
# importação do tempo de extração da linha
dftemposlinha = pd.read_excel(dados, dtype={'idLin': int, 'lin': int},
                              sheet_name="temposLin")
dftemposlinha.set_index('idLin', inplace=True)
temposlin = dftemposlinha.to_dict()['lin']
ltlin = dftemposlinha['lin'].values.tolist()

# importação do tempos de extração da posição
dftempospos = pd.read_excel(dados, dtype={'idPos': int, 'pos': int},
                            sheet_name="temposPos")
dftempospos.set_index('idPos', inplace=True)
tempospos = dftempospos.to_dict()['pos']
ltpos = dftempospos['pos'].values.tolist()

```

Figura 45 - Parte 5 do código do documento Python "docplex_model.py".

Por último, procedeu-se à importação dos dados do documento Excel, "*sequencias.xlsx*". A estrutura deste documento, assim como a sua constituição foram descritas na página 55. Deste modo, como a Figura 46 ilustra, foram importadas as referências, as posições e o valor de FIFO esperado, para cada posição da sequência de extração.

```
94  # importação da sequência de extração das refências
95  dfseq = pd.read_excel(sequencias, dtype={'ord': int, 'refS': int},
96                        sheet_name=folhaseq)
97  dfseq.set_index('ord', inplace=True)
98  seq = dfseq.to_dict()['refS']
99
100 # importação das posições da sequência de extração
101 dfpossaida = pd.read_excel(sequencias, dtype={'id': int, 'ord': int},
102                            sheet_name=folhaseq)
103 dfpossaida.set_index('id', inplace=True)
104 psaida = dfpossaida.to_dict()['ord']
105
106 # importação da ordem de FIFO da sequência de extração
107 dfseqfifo = pd.read_excel(sequencias, dtype={'ord': int, 'seqfifo': int},
108                           sheet_name=folhaseq)
109 dfseqfifo.set_index('ord', inplace=True)
110 seqfifo = dfseqfifo.to_dict()['seqfifo']
111 pseqfifo = dfseqfifo['seqfifo'].values.tolist()
112
```

Figura 46 - Parte 6 do código do documento Python "docplex_model.py".

Assim, com as linhas de código apresentadas na Figura 41, Figura 42, Figura 43, Figura 44, Figura 45 e Figura 46, foi possível instanciar e importar todos os dados necessários à construção em linguagem Python, do modelo matemático anteriormente definido. É importante referir que, devido à forma como código foi definido, sempre que foi necessário alterar a origem dos dados das configurações ou das sequências, apenas se modificaram as linhas entre 5 e 7 da Figura 41.

Para se iniciar a construção do modelo de otimização, foi necessário declarar, através da função, *Model()*, da biblioteca Docplex, o problema denominado "*Shopstocker*". Esta definição pode ser observada na linha 114 da Figura 47.

Neste modelo, foram desenvolvidas quinze variáveis de decisão, necessárias à otimização combinatória e à avaliação de diferentes KPI's. As linhas de código da Figura 47 e Figura 48, ilustram a definição de cada uma, com auxílio das funções do Docplex. A primeira variável de resposta, X*ij*, igualmente declarada nos modelos anteriores, era do tipo binário, assumindo o valor de 1 se, e só se, o PC, *i*, fosse alocado à posição, *j*, da sequência de extração. A função, *binary_var_dict()*, permitiu guardar

os valores assumidos por esta variável numa estrutura de dados do tipo dicionário. Para esta função foi ainda necessário indicar os índices da variável, os *sets* nos quais esses índices iteravam e, por fim, o nome de apresentação da variável após a execução do modelo. Da mesma forma, foi criada uma outra variável do tipo inteiro, Z*lp*, necessária à correta atribuição dos PC às posições da sequência de extração. Neste caso, foi utilizada a função, *integer_var_dict()*, do Docplex e, tal como na variável anterior, especificaram-se os índices, os *sets* e o nome final da variável em questão. As variáveis até aqui descritas correspondem ao modelo MPA.

```
113  # definição do problema
114  opt_model = cpx.Model(name='Shopstocker')
115
116  # variável binária x
117  var_x = opt_model.binary_var_dict([(i, j) for i in N for j in S],
118                                    name='x')
119  # variável inteira z
120  var_z = opt_model.integer_var_dict([(a, p) for a in L for p in P],
121                                     name='z')
122  # variável inteira fifo
123  var_fifo = opt_model.integer_var_dict([(i, j) for i in N for j in S],
124                                        name='fifo')
125  # variável binária fifo na posição j
126  var_fifo_j = opt_model.binary_var_dict([j for j in S], name='fifo_j')
127  # variável contínua %fifo
128  var_perc_fifo = opt_model.continuous_var(lb=0, name='%fifo')
129  # variável contínua complementar da %fifo
130  var_comp_fifo = opt_model.continuous_var(lb=0, name='comp_%fifo')
```

Figura 47 - Parte 7 do código do documento Python "docplex_model.py".

As quatro variáveis seguintes foram definidas para avaliar o KPI relativo ao cumprimento do FIFO que permitiram construir o modelo MF. A primeira, fifo*ij*, visível nas linhas 123 e 124 da Figura 47, foi definida com a mesma função do Docplex da variável anterior. Esta variável assumiu o valor do FIFO do PC, *i*, se e só se, esse PC fosse alocado a alguma posição, *j*. Desta forma, todos os PC selecionados pelo modelo, tiveram o seu valor de FIFO guardado nesta variável. Por outro lado, foi criada uma variável binária, *fifo_j*, utilizando a mesma função que na variável, X*ij*. Esta nova variável assumiu o valor de 1 caso o valor de FIFO do PC, *i*, fosse igual ao FIFO esperado na posição, *j*, e 0, no caso contrário. Por fim, as duas últimas variáveis representadas nas linhas 127 a 130 da Figura 47, foram definidas com a função, *continuous_var()*, onde se especificou, através do parâmetro, *lb=0*, que possuíam um limite inferior igual a 0. A variável, *%fifo*, assume o valor da percentagem de FIFO cumprida. Com o intuito de

minimizar este KPI na função objetivo, foi ainda desenvolvida a variável, *comp_%fifo*, que assume o valor complementar à percentagem de FIFO.

```
131  # variável inteira y
132  var_y = opt_model.integer_var_dict([(a, p) for a in L for p in P],
133                                     name='y')
134  # variável inteira mov
135  var_mov = opt_model.integer_var_dict([a for a in L], name='mov')
136  # variável inteira tot_mov
137  var_tot_mov = opt_model.integer_var(lb=0, name='tot_mov')
138  # variável inteira w
139  var_w = opt_model.integer_var_dict([(a, p) for a in L for p in P],
140                                     name='w')
141  # variável contínua tempo1
142  var_tempo1 = opt_model.continuous_var_dict([a for a in L],
143                                             name='tempo1')
144  # variável binária tempo2
145  var_tempo2 = opt_model.binary_var_dict([(a, p) for a in L for p in P],
146                                         name='tempo2')
147  # variável contínua tempo3
148  var_tempo3 = opt_model.continuous_var_dict([a for a in L],
149                                             name='tempo3')
150  # variável contínua tempolin
151  var_tempo_lin = opt_model.continuous_var_dict([a for a in L],
152                                                name='tempo_lin')
153  # variável contínua tot_tempo
154  var_tot_tempo = opt_model.continuous_var(lb=0, name='tot_tempo')
155
```

Figura 48 - Parte 8 do código do documento Python "docplex_model.py".

Para avaliar o KPI referente ao número de PC movimentados eliminando o efeito de duplicação, descrito mais adiante, foram desenvolvidas três variáveis de decisão, ilustradas na Figura 48. A primeira, Y*lp*, foi construída com a função, *integer_var_dict()*, do Docplex e permitiu guardar nos índices, linha e posição, o valor da posição do PC, *i*, caso a variável, X*ij*, assumisse o valor de 1. Por outro lado, esta variável assumiria o valor de 0, caso, X*ij*, também fosse 0. Por sua vez, a variável, mov*l*, assumiu para cada linha, a posição mais alta selecionada para extração. Por fim, a variável, tot_mov, recebeu o valor do somatório de todas as variáveis, mov*l*. Estas duas variáveis do tipo inteiro, foram implementadas igualmente com auxílio da função, *integer_var_dict()*, do Docplex.

As restantes seis variáveis presentes na Figura 48 foram desenvolvidas de modo a avaliar o tempo de extração sem o citado efeito de duplicação. Assim, na primeira variável do tipo inteiro, W*lp*, guardou-se, o valor do tempo de extração da posição do PC, *i*, caso esse fosse selecionado. A segunda variável

contínua, tempo1*l*, implementada com a função, *continuous_var_dict()*, do Docplex, assumiu, para cada linha, o tempo da posição mais elevada selecionada para a extração. Da mesma forma, a variável de resposta, tempo2*lp*, do tipo binário, assumiu o valor de 1, no caso da variável, W*lp*, assumir um valor diferente de 0. A quarta variável desenvolvida para este KPI, tempo3*l*, recebeu, para cada linha, o valor do somatório do tempo de extração dos PC selecionados dessa linha. A penúltima variável, temp_lin*l*, armazenou o valor total, para cada linha do tempo de extração dos seus PC selecionados. Esta variável, recebeu assim o somatório das variáveis, tempo1*l* e tempo2*l*. Por fim, a variável, tot_tempo, guardou o somatório de todas as variáveis, tempo*l*, refletindo assim, o tempo total de extração dos PC selecionados numa determinada solução. Estas três últimas variáveis foram igualmente definidas com a função, *continuous_var_dict()*, do Docplex.

A declaração destas variáveis extra relativas às medidas de desempenho do número de PC movimentados e ao tempo de extração permitiu desenvolver o modelo MTFM.

Após a definição das variáveis de resposta necessárias, procedeu-se ao desenvolvimento do modelo matemático definido na subsecção 4.5.

```
156  # função objetivo 1 - custo da linha * custo da posição
157  objetivo1 = opt_model.sum(var_x[i, j] * lin[i] * pos[i]
158                            for i in N for j in S)
159
160  # função objetivo 2 - complementar da % FIFO vs tempo de extração (inicial)
161  objetivo2 = var_comp_fifo * 0.9912 + (opt_model.sum(var_x[i, j] * t_ext[i]
162                              for i in N for j in S)) * 0.0088
163
164  # função objetivo 2 - curva fronteira Pareto
165  objetivo2 = opt_model.sum(var_x[i, j] * t_ext[i] for i in N for j in S)
166  objetivo2 = var_comp_fifo
167
168  # função objetivo 3 - tempo de extração (real) vs complementar da % FIFO vs
169  # PC movimentados
170  objetivo3 = var_tot_tempo * 0.0088 + var_comp_fifo * 0.9912 + var_tot_mov \
171              * 0.9912
172
```

Figura 49 - Parte 9 do código do documento Python "docplex_model.py".

Tal como a Figura 49 mostra, foram definidas, entre as linhas 156 e 171 três funções objetivo. Cada uma destas funções foi submetida ao modelo de otimização de forma independente e exclusiva, selecionando-se em cada momento, a que melhor se enquadrava com a experiência a realizar.

A primeira função, *objetivo1*, executou o somatório do produto entre a variável, X*ij*, e o custo de alocação do PC, *i*. Por simplificação deste modelo, o custo de alocação foi definido como a multiplicação entre a linha e a posição onde o PC estava armazenado. Esta função objetivo, igual à desenvolvida nos modelos implementados com as bibliotecas *Pulp* e *Pyomo*, corresponde à utilizada pelo MPA.

Por sua vez, na segunda função objetivo foram incorporados dois objetivos concorrentes: complementar da percentagem de FIFO cumprida e tempo de extração do PC, *i*. É importante referir que este tempo de extração contabiliza duplamente PC extraídos da mesma linha. A título de exemplo, se de uma determinada linha saírem os PC das posições 3 e 5, a função objetivo contabilizará o somatório dos tempos de extração das posições 3 e 5, respetivamente. No entanto, como o fluxo ao longo da linha é feito por gravidade, à medida que o PC da posição 3 vai sendo extraído, os restantes PC da linha vão-se igualmente aproximando da extremidade de saída. Por esta razão, quando o segundo PC selecionado nesta linha tiver que sair, não estará na posição inicial 5, mas sim na posição 2. Esta função objetivo corresponde á do modelo MF implementado.

Por fim, a terceira função objetivo corresponde ao modelo MTFM, em que se implementaram novas variáveis e restrições com o intuito de eliminar o efeito de duplicação anteriormente descrito. Assim, nesta função objetivo garantiu-se que a seleção dos melhores PC era feita procurando avaliar três KPI's simultaneamente: tempo real de extração, percentagem de FIFO cumprida e o número real de PC movimentados.

```
173     # restrição 1 cada posição j da sequência é preenchida uma e uma só vez
174     for j in S:
175         opt_model.add_constraint(opt_model.sum(var_x[i, j] for i in N) == 1)
176
177     # retrição 2 cada PC i é atribuido no máximo uma vez a uma posição j
178     for i in N:
179         opt_model.add_constraint(opt_model.sum(var_x[i, j] for j in S) <= 1)
180
181     # restrição 3 a referência do PC i alocado é igual à pretendida na posição j
182     for i in N:
183         for j in S:
184             opt_model.add_constraint(var_x[i, j] * ref[i] == var_x[i, j] * seq[j])
185
186     # restrições 4 e 5 PC invalidados por impedimento físico
187     for i in N:
188         for j in S:
189             opt_model.add_if_then(var_x[i, j] == 1, var_z[lin[i], pos[i]] ==
190                                   psaida[j])
191     for a in L:
192         for p in range(1, N_pos):
193             opt_model.add_constraint(var_z[a, p] <= var_z[a, p + 1])
194
```

Figura 50 - Parte 10 do código do documento Python "docplex_model.py".

Do mesmo modo, procedeu-se ao desenvolvimento das restrições definidas no modelo matemático da subsecção 4.5. Assim, a primeira restrição, declarada nas linhas 174 e 175 da Figura 50, assegurou que cada posição da sequência seria preenchida uma única vez. A segunda restrição, permitiu especificar que cada PC, *i*, apenas era passível de ser alocado uma vez para o conjunto de todas as posições, *j*, da sequência de extração. Por fim, a restrição desenvolvida e apresentada nas linhas 182 a 184 da Figura 50, garantiu que a referência do PC, *i*, alocado era igual à pretendida na posição, *j*, da sequência de extração. A função, *add_constraint()*, da biblioteca Docplex, permitiu adicionar ao modelo as três restrições acima descritas.

Ao contrário das bibliotecas Pulp e Pyomo anteriormente apresentadas, o Docplex possui uma função que permite implementar diretamente a restrição (5) do modelo matemático definido na subsecção 4.5. A função, *add_if_then(A, B)*, obriga o modelo a cumprir *B* sempre que *A* se verifique. Esta descoberta possibilitou que o modelo matemático definido fosse integralmente implementado com a biblioteca Docplex. As restrições apresentadas entre as linhas 186 a 193 da Figura 50, ilustram as restrições (5) e (6) do modelo matemático. Na primeira, foi assegurado que sempre que a variável, X*ij*, assumiu o valor 1, a posição, *j*, foi guardada na variável, Z*lp*. Os índices, *l*, e, *p*, representam, respetivamente, a linha e a posição do PC, *i*, alocado. Por outro lado, na restrição das linhas 191 a 193, foi assegurado que o valor da variável, Z*lp*, fosse sempre menor ou igual ao valor de Z, na posição

seguinte da mesma linha. Desta forma, assegurou-se que nenhum PC, extraído por ser um impedimento físico ao PC, *i*, pudesse ser alocado a uma posição, *j*, da sequência de extração, superior à do PC, *i*. Com este conjunto de restrições terminou-se a implementação do modelo MPA.

```
195     # restrições 6, 7, 8, 9 e 10 KPI % de FIFO
196     for i in N:
197         for j in S:
198             opt_model.add_constraint(var_fifo[i, j] == var_x[i, j] * fifo[i])
199     for i in N:
200         for j in S:
201             opt_model.add_if_then(pseqfifo[j-1] * var_x[i, j] !=
202                                   var_fifo[i, j], var_fifo_j[j] == 0)
203     opt_model.add_constraint(var_perc_fifo ==
204                              (opt_model.sum(var_fifo_j[j] for j in S) / N_seq) * 100)
205     opt_model.add_constraint(var_comp_fifo == 100 - var_perc_fifo)
206     opt_model.add_constraint(var_perc_fifo >= 0)
207
```

Figura 51 - Parte 11 do código do documento Python "docplex_model.py".

De forma a tornar o modelo mais completo, procurou-se incorporar no mesmo, o KPI referente à percentagem de FIFO cumprida pelos PC selecionados para a extração. A restrição implementada entre as linhas 196 a 198 da Figura 51, permitiu guardar na variável, fifo*ij*, anteriormente descrita, o valor do FIFO associado ao PC, *i*, alocado à posição, *j*. Esta restrição garantiu ainda que, para todos os PC não alocados, o valor da variável, fifo*ij,* fosse 0. A segunda restrição alusiva ao cálculo da percentagem de FIFO, foi definida nas linhas 199 a 202 da Figura 51. Esta restrição, declarada com auxílio da função, *add_if_then()* do Docplex, permitiu atribuir o valor de 0 à variável, *fifo_j*, a todos os PC, *i*, alocados, no caso do seu valor de FIFO ser diferente do requerido na posição, *j*. Desta forma, apenas nas posições da sequência de extração em que o FIFO era cumprido, a variável binária, *fifo_j*, assumiu o valor de 1. Na terceira restrição ilustrada nas linhas 203 e 204 da Figura 51, calculou-se a percentagem de FIFO cumprida. Tal como a Figura citada demonstra, à variável, *perc_fifo*, atribuiu-se o valor da razão entre o somatório da variável, *fifo_j*, em todas as posições, *j*, e o total de posições da sequência de extração. A posterior multiplicação pelo valor de 100, permitiu guardar em percentagem o valor anteriormente calculado. De seguida, com o intuito de uniformizar a índole de todas as funções objetivo, desenvolveu-se a restrição ilustrada na linha 205 da mesma Figura, em que se definiu a variável, *comp_fifo*, como sendo o valor complementar da percentagem de FIFO cumprida. Desta forma, com esta nova variável, foi possível minimizar todas as funções objetivo presentes na Figura 49. A última restrição desenvolvida

relativa ao cálculo da percentagem de FIFO, permitiu definir um limite mínimo da percentagem de FIFO exigida em cada experiência realizada. Com este conjunto de restrições concluiu-se a implementação do modelo MF, a variável relativa ao KPI da percentagem de FIFO cumprida e o parâmetro do tempo de extração com efeito de duplicação, eram os fatores chave para a otimização.

```
208     # restrições 11, 12 e 13 KPI PC movimentados
209     for i in N:
210         opt_model.add_constraint(opt_model.sum(var_x[i, j] for j in S) *
211                                  pos[i] == var_y[lin[i], pos[i]])
212     for a in L:
213         opt_model.add_constraint(var_mov[a] == opt_model.max(var_y[a, p]
214                                                              for p in P))
215     opt_model.add_constraint(var_tot_mov == opt_model.sum(var_mov[a]
216                                                           for a in L))
217
```

Figura 52 - Parte 12 do código do documento Python "docplex_model.py".

Na última fase de desenvolvimento do modelo, com o intuito de eliminar o efeito de duplicação anteriormente descrito, procurou-se desenvolver o modelo MTFM, no qual o tempo total de extração e o número de PC movimentados já se encontravam devidamente corrigidos.

De forma a analisar o KPI referente ao número de PC movimentados sem o efeito de duplicação, desenvolveram-se três restrições ilustradas na Figura 52. A primeira restrição deste conjunto permitiu guardar na variável, Y*lp*, o valor da posição do PC, *i*, caso este tivesse sido alocado a alguma posição, *j*, da sequência de extração. Deste modo, apenas nos índices linha e posição dos PC, *i*, selecionados, a variável, Y*lp*, assumiu o valor da posição, ficando as restantes com o valor de 0. Na segunda restrição, com o intuito eliminar o efeito de duplicação existente quando se extraí mais do que um PC da mesma linha, optou-se por guardar, na variável, mov*l*, a posição mais alta utilizada em cada linha. A título de exemplo, se de uma determinada linha fossem retirados 3 PC das posições 1, 5, e 15, a variável, mov*l*, apenas guardaria o valor de 15 no índice da linha em questão. Para implementar estra restrição utilizou-se a função, *max()*, do Docplex, que procurou, para cada linha, o valor de, Y*lp*, mais elevado. Por outro lado, nas linhas das quais nenhum PC foi selecionado, o valor de todas as variáveis, Y*lp*, era de 0, razão pela qual, a variável, mov*l*, nessa linha assumiu o valor de 0. Assim, com a aplicação desta restrição, anulou-se o efeito de duplicação e, consequentemente, a contabilização dos PC movimentados tornou-se numa medida exata. Por fim, para contabilizar o total de PC movimentados, desenvolveu-se a terceira restrição deste conjunto onde a variável, tot_mov, recebeu o somatório de todas as variáveis, mov*l*. Esta restrição foi implementada com o auxílio da função, *sum()*, anteriormente descrita. Estas três restrições

foram adicionadas ao modelo de otimização através da função, *add_constraint()*, do Docplex. Deste modo, com este conjunto de restrições, foi possível minimizar, na função objetivo, a variável, tot_mov, cujo valor contempla o número real de PC movimentados, sem o efeito de duplicação.

```
218     # restrições 14, 15, 16, 17, 18, 19 e 20 KPI tempo de extração
219     for i in N:
220         opt_model.add_constraint(opt_model.sum(var_x[i, j] for j in S) * tpos[i]
221                                  == var_w[lin[i], pos[i]])
222     for a in L:
223         opt_model.add_constraint(var_tempo1[a] == opt_model.max(var_w[a, p]
224                                                  for p in P))
225     for a in L:
226         for p in P:
227             opt_model.add_if_then(var_w[a, p] != 0, var_tempo2[a, p] == 1)
228     for a in L:
229         opt_model.add_constraint(var_tempo3[a] == opt_model.sum(var_tempo2[a, p]
230                                                  for p in P) * temposlin[a])
231     for a in L:
232         opt_model.add_constraint(var_tempo_lin[a] == var_tempo1[a] +
233                                  var_tempo3[a])
234     opt_model.add_constraint(var_tot_tempo == opt_model.sum(var_tempo_lin[a]
235                                                  for a in L))
236
```

Figura 53 - Parte 13 do código do documento Python "docplex_model.py".

As últimas sete restrições desenvolvidas neste modelo permitiram calcular o tempo de extração real, isto é, o tempo de extração sem o efeito de duplicação. Este conjunto de restrições está ilustrado na Figura 53. A restrição 14, primeira deste conjunto, permitiu guardar na variável, W*lp*, o tempo de extração do PC, *i*, da sua posição, caso este fosse selecionado. Para os restantes PC, *i*, não escolhidos para a solução, a variável, W*lp*, assumiu o valor de 0. Esta restrição foi implementada com auxílio da função, *sum()*, anteriormente descrita. De seguida, para eliminar o efeito de duplicação, guardou-se na variável, tempo1*l*, o valor de extração mais elevado. Deste modo, e tal como a restrição 15 permite perceber, se numa determinada linha fossem retirados três PC com os tempos de extração de 8, 40 e 75 segundos, a variável, W*lp*, apenas receberia o tempo mais elevado e não o somatório dos três. Desta forma, e com a utilização da função, *max()*, do Docplex, eliminou-se o efeito de duplicação de tempos. Uma vez que, o tempo total de saída contempla o somatório do tempo de extração do PC da respetiva linha e do tempo de movimentação da linha até à zona de *output*, foi necessário desenvolver as restrições 16 e 17, para calcular esta última parcela. Para este efeito, utilizou-se na restrição 16, a função, *add_if_then()*, que permitiu ativar a variável binária, tempo2*lp*, sempre que a variável, W*lp*, assumisse

um valor diferente de 0. Por sua vez, a restrição 17 permitiu guardar na variável, tempo3*l*, o somatório da multiplicação do tempo de extração da respetiva linha pelo número de PC movimentados na mesma, presentes na variável, tempo2*lp*. Assim, as variáveis, tempo1*l*, e, tempo3*l*, armazenavam, respetivamente, os tempos de extração de todos os PC selecionados, das posições e das linhas, em que estavam armazenados. A restrição 18 realizou assim o somatório, para cada linha, dos tempos presentes nas duas variáveis citadas, ficando a variável, tempo_lin*l*, com o tempo total de extração dos PC selecionados de cada linha de armazenamento. A restrição 19, por sua vez, possibilitou guardar na variável, tot_tempo, o somatório de todas as variáveis, tempos_lin*l*, correspondentes ao tempo total de extração de todos os PC do Shopstocker. Neste conjunto de restrições, à exceção da restrição 16, todas as restantes foram implementadas com a utilização da função, *add_constraint()*, da biblioteca Docplex.

```
237  # KPI do tempo de extração (sem correção de posições)
238  opt_model.add_kpi(opt_model.sum(var_x[i, j]*t_ext[i] for i in N for j in S),
239                    publish_name="Tempo Extração (sem correção) em segundos")
240  # KPI do tempo de extração (com correção de posições)
241  opt_model.add_kpi(var_tot_tempo, publish_name="Tempo Extração em segundos")
242
243  # KPI da % de FIFO cumprida
244  opt_model.add_kpi(var_perc_fifo, publish_name="Percentagem de FIFO")
245
246  # KPI do PC movimentados (sem correção de posições)
247  opt_model.add_kpi(opt_model.sum(var_x[i, j]*pos[i] for i in N for j in S),
248                    publish_name="Para-choques Movimentados (sem correção)")
249  # KPI do PC movimentados (com correção de posições)
250  opt_model.add_kpi(var_tot_mov, publish_name="Para-choques Movimentados")
251
```

Figura 54 - Parte 14 do código do documento Python "docplex_model.py".

Após uma nova investigação da documentação da biblioteca Docplex, percebeu-se que, através da função, *add_KPI()*, se poderiam analisar os valores dos KPI's desenvolvidos. Desta forma, e como a Figura 54 permite perceber, foram implementados cinco KPI's, calculados durante o processo de otimização. O primeiro, exposto nas linhas 238 e 239 da referida Figura, apresentou o somatório dos tempos de extração de todos os PC selecionados com o efeito de duplicação. O KPI seguinte, apresentou o valor do tempo de extração, obtido com a variável, *tot_tempo*, exibindo por isso, o tempo de extração real, em cada solução. O terceiro avaliou a percentagem de FIFO cumprida, através do valor da variável, *perc_fifo*, anteriormente descrita. Por fim, desenvolveram-se dois KPI's relativos ao número de PC movimentados. O quarto KPI apresentou assim, através do somatório das posições dos PC selecionados,

o total de movimentados com o efeito de duplicação. No quinto e último KPI desenvolvido, contabilizaram-se apenas as posições selecionadas mais elevadas de cada linha, através da variável, *tot_mov*, o que permitiu perceber o número total de PC movimentados, já devidamente corrigido. O parâmetro, *publish_name=*, adicionado a cada KPI, permitiu definir o nome pelo qual este seria apresentado no *output* do modelo.

Com as funções objetivo e as restrições do modelo devidamente formuladas e traduzidas para a linguagem Python, procedeu-se à implementação das funções de resolução do problema. Na linha 253 da Figura 55, aplicou-se a função, *minimize()*, que definiu a índole da função objetivo nela contida.

```
252  # minimização da função objetivo
253  opt_model.minimize(objetivo3)
254
255  # estabelece um limite de 250 segundos para a execução da otimização
256  opt_model.parameters.mip.tolerances.mipgap = 0.05
257  opt_model.set_time_limit(250.0)
258
259  # resolve o problema com o algoritmo CPLEX
260  solucao = opt_model.solve(log_output=True)
261
262  # exibe os resultados da otimização
263  print(solucao)
264  solval_x = solucao.get_value_dict(var_x, keep_zeros=False)
265
266  print("=-=" * 26)
267  print(opt_model.get_solve_details())
268
269  print("=-=" * 26)
270  # reporta os valores dos KPI's definidos
271  opt_model.report_kpis()
272
```

Figura 55 - Parte 15 do código do documento Python "docplex_model.py".

A construção deste problema de otimização, e a sua resolução através do algoritmo CPLEX, teve como principal objetivo a obtenção de uma solução ótima. No entanto, devido às restrições existentes na empresa, o tempo de execução da otimização, e a consequente seleção dos melhores PC para cada sequência de extração, não poderia exceder os 300 segundos. Por essa razão, definiu-se na linha 257 da Figura 55, que uma solução inteira viável, teria que ser exibida ao fim de 250 segundos de otimização, aos quais acrescem 50 segundos de pré-processamento do modelo. Desta forma, se numa determinada experiência o tempo necessário para a resolução do problema fosse superior ao estipulado, o modelo apresentaria, não a solução ótima, mas sim a melhor solução calculada no intervalo de tempo limite. Por

outro lado, foi também definido um GAP de 5% entre a melhor solução inteira encontrada e o valor do melhor nodo restante. Caso o valor do GAP fosse inferior a 5%, a otimização seria interrompida, e a melhor solução inteira encontrada seria apresentada. Através do *output* gerado pelo modelo foi possível avaliar em cada momento de que tipo de solução se tratava.

Com os dados, o modelo e os parâmetros necessários devidamente implementados, aplicou-se a função, *solve()*, que resolveu este problema de programação linear inteira mista. O parâmetro, *log_output=True*, especificou que se pretendiam obter os parâmetros da execução da otimização no *output* do modelo. Este *output* foi guardado num objeto categorizado como, *solucao*. A função, *print()*, do Python exposta na linha 263 da Figura 55, permitiu posteriormente apresentar a solução obtida.

Para que fosse possível trabalhar os valores otimizados da variável, X*ij*, escreveu-se a linha de código 264, da Figura 55. A função, *get_value_dict()*, permitiu guardar num objeto categorizado como, *solval_x*, os valores da variável, X*ij*. O parâmetro, *keep_zeros=False*, estabeleceu que apenas os valores diferentes de zero seriam armazenados.

Da mesma forma, para avaliar o tipo de solução obtida e o tempo de execução do modelo de otimização, empregou-se a função, *get_solve_details()*, do Docplex. Por fim, a função, *report_kpis()*, presente na linha 271 da Figura 55, foi invocada para que os valores dos KPI's definidos, fossem exibidos no *output* do modelo.

```
273  # análise pós-otimização
274
275  # definição de listas
276  ri = []
277  rlin = []
278  rpos = []
279  rfifo = []
280  jfifo = []
281  vetorlinhas = []
282  vetorPR = []
283
284  # associação de cada valor pretendido à lista
285  for c, (k, values) in enumerate(solval_x):
286      ri.append(k)
287      rlin.append(lin[k])
288      rpos.append(pos[k])
289      rfifo.append(fifo[k])
290      jfifo.append(values)
291
```

Figura 56 - Parte 16 do código do documento Python "docplex_model.py".

De forma a obter o valor dos três KPI's devidamente corrigidos diretamente no output do documento, "*docplex_model.py*", independentemente do modelo utilizado, definiram-se algumas funções executadas após a otimização. A primeira etapa, ilustrada na Figura 56, consistiu na declaração de listas, necessárias para o posterior tratamento de dados. As listas foram definidas pela seguinte ordem: *ri*, destinada a armazenar todos os valores de, *i*, identificadores dos PC selecionados; *rlin*, que armazenou para cada PC, *i*, selecionado, a sua linha de armazenamento; *rpos*, que recebeu as posições de todos os PC, *i*, selecionados; *rfifo*, que guardou os valores de FIFO de todos os PC, *i*, selecionados; *jfifo*, que armazenou para cada PC, *i*, escolhido, o valor de FIFO associado à posição, *j*; *vetorlinhas*, que recebeu de forma única todas as linhas das quais foram retirados PC e, por fim, *vetorPR*, destinado a guardar todas as posições devidamente corrigidas dos PC, selecionados. As linhas 285 a 290 da Figura 56, ilustram a atribuição dos valores anteriormente descritos, às cinco primeiras listas. Para esta atribuição, utilizou-se a função, *enumerate()*, do Python que permitiu iterar nos valores e chaves do dicionário nele contido. No caso em estudo, a função foi aplicada ao objeto declarado como, *solval_x*, que continha os valores otimizados da variável, X*ij*.

```
292     # cálculo do número de PC movimentados
293     for j in S:
294         if rlin[j-1] not in vetorlinhas:
295             vetorlinhas.append(rlin[j-1])
296
297     for c in range(0, len(vetorlinhas)):
298         vetorPR.append(0)
299
300     linha = 1
301     c = 0
302     for linha in vetorlinhas:
303         c += 1
304         for j in S:
305             if rlin[j-1] == linha:
306                 if rpos[j-1] > vetorPR[c-1]:
307                     vetorPR[c-1] = rpos[j-1]
308
309     print('=-=' * 26)
310     pc_mov = 0
311     for c in vetorPR:
312         pc_mov += c
313     print(f'*  KPI: Para-Choques Movimentados       = {pc_mov}')
314     print(f'*  KPI: Para-choques Impedimento Físico = {pc_mov-N_seq}')
315
```

Figura 57 - Parte 17 do código do documento Python "docplex_model.py".

Com as listas anteriormente criadas, procedeu-se ao cálculo do número de PC movimentados sem o efeito de duplicação anteriormente descrito. Tal como a Figura 57 ilustra, armazenaram-se de forma única, na lista, *vetorlinhas*, todas as linhas das quais um qualquer PC, *i*, foi selecionado. É importante referir que, mesmo que de uma determina linha de armazenamento saíssem mais do que um PC, a lista, *vetorlinhas*, apenas a guardaria essa linha uma única vez.

Por sua vez, as linhas 297 e 298 da Figura 57, atribuíram à lista, *vetorPR*, o valor de zero para cada posição correspondente a uma linha guardada na lista, *vetorlinhas*. Assim, para cada N linhas diferentes armazenadas na lista, *vetorlinhas*, seriam guardados N zeros no vetor, *vetorPR*.

Para eliminar o efeito de duplicação anteriormente descrito, implementaram-se as linhas de código 300 a 307 da Figura 57. Para cada posição da lista, *vetorPR*, inicialmente com o valor de zero, guardou-se a posição mais elevada, entre todos os PC, *i*, selecionados da linha correspondente. Com esta estratégia, apenas a posição escolhida mais elevada dentro de cada linha, ficava registada na lista, *vetorPR*. As restantes posições da referida linha seriam omitidas, evitando-se o efeito de duplicação de posições.

Por último, com as posições devidamente tratadas, executou-se o somatório todas as posições guardadas na lista, *vetorPR*. Deste modo, foi possível perceber o valor exato de PC movimentados em cada situação e, dentro desses, quantos foram movimentados por impedimento físico à extração dos demais.

```
316     # cálculo da % de FIFO cumprida
317     aux_fifo = dict(zip(jfifo, rfifo))
318     aux_fifo2 = sorted(aux_fifo.items())
319     fifo_sol = []
320     for j in S:
321         fifo_sol.append(aux_fifo2[j-1][1])
322
323     print('=-=' * 26)
324     cont_fifo = 0
325     for j in S:
326         if fifo_sol[j-1] == pseqfifo[j-1]:
327             cont_fifo += 1
328     perc_fifo = (cont_fifo/N_seq) * 100
329     print(f'*  KPI: Percentagem de FIFO = {perc_fifo:.2f}%')
330
```

Figura 58 - Parte 18 do código do documento Python "docplex_model.py".

Apesar da percentagem de FIFO ser reportada corretamente através do KPI definido durante a otimização, optou-se por realizar um segundo cálculo, com o intuito de validar a percentagem obtida. Ao contrário do KPI descrito e apresentado na Figura 54, a percentagem de FIFO apresentada na Figura 58 foi calculada com os resultados da otimização e não, durante a execução da mesma.

A primeira etapa consistiu na criação de um dicionário, através da função, *dict()*, do Python. A linha 317 da Figura 58, mostra a aplicação da função, *zip()*, que atribuiu os valores da lista, *jfifo*, às chaves dos dicionários, e associou a cada um, o valor de FIFO guardado no vetor, *rfifo*. Posteriormente, ordenou-se o dicionário criado pelos seus valores de chave. Assim, desenvolveu-se um objeto, denominado *aux_fifo2*, com os valores de FIFO selecionados para cada posição, *j*, devidamente ordenados. Posteriormente, criou-se uma nova lista, *fifo_sol*, que recebeu apenas os valores de FIFO, já devidamente ordenados.

Por fim, para avaliar a percentagem de FIFO cumprida, verificou-se se o valor guardado em cada posição da lista criada, *fifo_sol*, correspondia ao requerido na mesma posição do vetor, *pseqfifo*, importado diretamente do ficheiro Excel. Deste modo, através do código visível na Figura 58, foi possível calcular a percentagem de FIFO cumprida, com base nos valores otimizados da variável, X*ij*.

```
331  # cálculo do tempo de extração dos PC
332  print('=-=' * 26)
333
334  tot_ext = 0
335  for i in vetorPR:
336      tot_ext += ptpos[i-1]
337
338  for a in rlin:
339      tot_ext += temposlin[a]
340
341  temp_ref = (80 * N_seq)
342
343  print(f'*  KPI: Tempo de Referência de Extração (Tack-Time) = '
344        f'{temp_ref:.2f} s = {temp_ref/60:.2f} min = '
345        f'{temp_ref/(60*60):.2f} h')
346
347  print(f'*  KPI: Tempo de Extração                 = '
348        f'{tot_ext:.2f} s = {tot_ext/60:.2f} min = '
349        f'{tot_ext/(60*60):.2f} h')
350
```

Figura 59 - Parte 19 do código do documento Python "docplex_model.py".

O último KPI analisado foi o tempo total de extração de todos os PC selecionados. Tal como o número de PC movimentados, o KPI desenvolvido e ilustrado na Figura 54, contém o efeito de duplicação anteriormente explicado. Para anular esse efeito, construíram-se as linhas de código expostas na Figura 59. Deste modo, para o cálculo deste KPI, procurou-se tirar proveito da lista anteriormente definida, *vetorPR*. Assim, com as posições selecionadas mais elevadas de cada linha, adicionou-se ao objeto, *tot_ext*, o tempo de extração de cada PC, associado às posições existentes no referido vetor. Desta forma, apenas foi contabilizado o tempo do PC de posição mais elevada em cada linha, evitando-se o efeito de duplicação. Por outro lado, para cada PC selecionado, foi adicionado o seu tempo de extração da linha. Neste caso, não existe efeito de duplicação, uma vez que se considerou que, o percurso de um PC ao longo do corredor do Shopstocker não influência o tempo de extração de outro PC da mesma linha.

Por fim, com o intuito de avaliar e comparar o tempo total de extração, desenvolveu-se o um tempo de referência, *temp_ref*, visível na linha 341 da Figura 59, que apresentou o tempo total de extração, caso todos os PC selecionados cumprissem rigorosamente os 80 segundos de intervalo, definidos como *takt time*.

```
351    # construção gráfica do Shopstocker
352
353    # geração das posições inexistentes
354    pos_retiradas = range(16, 26)
355    lin_retiradas = range(44, 46)
356    glin_ret = []
357    gpos_ret = []
358    for a in lin_retiradas:
359        for p in pos_retiradas:
360            glin_ret.append(a)
361            gpos_ret.append(p)
362
363    # geração das linhas dos pisos inferior (1) e superior (2)
364    plin_1 = []
365    plin_2 = []
366    for i in N:
367        if i <= (N_pc/2):
368            plin_1.append(plin[i-1])
369        else:
370            plin_2.append(plin[i-1])
371
```

Figura 60 - Parte 20 do código do documento Python "docplex_model.py".

A última etapa do algoritmo desenvolvido, consistiu na análise gráfica dos PC selecionados, permitindo inferir de forma automática, a sua localização ao longo do Shopstocker.

A Figura 60 ilustra, numa primeira fase, a criação de duas listas, *glin_ret*, e, *gpos_ret*, com o objetivo de receberem, respetivamente, as linhas e posições correspondentes à zona de armazenamento inexistente das linhas de pré-sequenciamento. Por outro lado, com o objetivo de apresentar graficamente o Shopstocker divido pelos dois pisos, geraram-se as listas, *plin_1*, e, *plin_2*, visíveis na mesma Figura, destinadas a guardar as linhas dos pisos inferior e superior, respetivamente. Da mesma forma, os vetores, *ppos_1*, e, *ppos_2*, expostos na Figura 61 foram gerados com o propósito de armazenarem as posições de todos os PC do piso inferior e superior, respetivamente.

```
372  # geração das posições dos pisos inferior (1) e superior (2)
373  ppos_1 = []
374  ppos_2 = []
375  for i in N:
376      if i <= (N_pc/2):
377          ppos_1.append(ppos[i-1])
378      else:
379          ppos_2.append(ppos[i-1])
380
381  # geração dos PC dos pisos inferior (1) e superior (2)
382  ri_1 = []
383  ri_2 = []
384  for i in N:
385      if i <= (N_pc/2):
386          ri_1.append(i)
387      else:
388          ri_2.append(i)
389
```

Figura 61 - Parte 21 do código do documento Python "docplex_model.py".

A Figura 60 ilustra ainda, nas suas linhas 381 a 388, a criação de duas listas, *ri_1*, e, *ri_2*, que armazenaram, de entre todos os PC, *i*, selecionados, os que pertenciam ao primeiro e ao segundo piso, respetivamente.

Com auxílio do vetor, *ri*, anteriormente descrito, foi possível guardar, as posições correspondentes aos PC selecionados dos pisos inferior, *rpos_1*, e superior, *rpos_2*. As linhas de código apresentadas na Figura 62, revelam ainda a criação das listas, *rlin_1*, e, *rlin_2*, destinadas a armazenar as linhas dos pisos inferior e superior, respetivamente, dos PC escolhidos para a sequência de extração.

Com este conjunto de etapas, conseguiram-se gerar no formato adequado, os dados necessários para a construção gráfica do Shopstocker e dos PC selecionados.

```
390  # geração das posições selecionadas dos pisos inferior (1) e superior (2)
391  rpos_1 = []
392  rpos_2 = []
393  for i in ri:
394      if lin[i] <= (N_lin/2):
395          rpos_1.append(pos[i])
396      else:
397          rpos_2.append(pos[i])
398
399  # geração das linhas selecionadas dos pisos inferior (1) e superior (2)
400  rlin_1 = []
401  rlin_2 = []
402  for i in ri:
403      if lin[i] <= (N_lin/2):
404          rlin_1.append(lin[i])
405      else:
406          rlin_2.append(lin[i])
407
```

Figura 62 - Parte 22 do código do documento Python "docplex_model.py".

Por sua vez, o código presente na permitiu ilustrar a implementação gráfica do Shopstocker com auxílio da biblioteca Matplotlib. Este módulo possui um conjunto de funções que permitem criar visualizações estáticas, dinâmicas ou interativas em linguagem Python. O estudo da documentação do Matplotlib, disponibilizada por Hunter, Dale, Firing, Droettboom, & Team (2012), permitiu inferir quais as funções e parâmetros que melhor se enquadravam neste projeto.

A função, *figure()*, presente na linha 409 da Figura 63, permitiu criar a imagem na qual os gráficos foram desenvolvidos. Posteriormente, procedeu-se ao desenvolvimento independente de dois gráficos, através da função, *subplot()*, correspondentes ao primeiro e segundo piso do Shopstocker, respetivamente. Tal como as linhas 411 a 420 da referida Figura demonstram, foram utilizados os vetores anteriormente definidos, contendo a informação relativa às linhas, *plin_1*, e às posições, *ppos_1*, do piso inferior. Os PC escolhidos, cuja informação ficou guardada nas listas, *rlin_1*, e, *rpos_1*, foram posteriormente adicionados, de forma a identificarem a região selecionada no conjunto de todas as posições do primeiro piso do armazém. Este procedimento foi repetido para o segundo piso, utilizando as respetivas listas. As diferentes funções utilizadas e identificadas na Figura 63 possuíam as seguintes funções: *plot()*, gerou um gráfico com os vetores nela contidos; *xlabel()*, formatou o eixo das abscissas do gráfico; *ylabel()*, formatou o eixo das ordenadas do gráfico; *title()*, definiu o título identificador do gráfico desenvolvido; *axis()*, formatou a escala dos eixos do gráfico e, por fim, *show()*, mostrou na imagem criada, o gráfico desenvolvido.

```
# construção da imagem
plt.figure()

# construção do piso inferior
plt.subplot(211)
plt.plot([ppos_1], [plin_1], '.k')
plt.plot([rpos_1], [rlin_1], '.y')
plt.xlabel(xlabel='<---- Posições -----', fontsize='small',
           verticalalignment='top', horizontalalignment='center')
plt.ylabel(ylabel='----- Linhas ---->', fontsize='small',
           verticalalignment='bottom', horizontalalignment='center')
plt.title(label="Shopstocker 1° Piso")
plt.axis([0, N_pos + 1, 44, 0])

# construção do piso superior
plt.subplot(212)
plt.plot([ppos_2], [plin_2], '.k')
plt.plot([rpos_2], [rlin_2], '.y')
plt.plot([gpos_ret], [glin_ret], '.w')
plt.axis([0, N_pos + 1, N_lin + 1, 43])
plt.xlabel(xlabel='<---- Posições -----', fontsize='small',
           verticalalignment='top', horizontalalignment='center')
plt.ylabel(ylabel='----- Linhas ---->', fontsize='small',
           verticalalignment='bottom', horizontalalignment='center')
plt.title(label="Shopstocker 2° Piso")
plt.show()

```

Figura 63 - Parte 23 do código do documento Python "docplex_model.py".

O programa desenvolvido, permitiu assim avaliar diretamente três KPI's, isto é, sem necessidade de cálculos posteriores à execução do mesmo, e gerou, para cada configuração, uma imagem gráfica do Shopstocker. Deste modo, com o programa implementado e apresentado nesta subsecção, puderam-se realizar diversas experiências com modelos desenvolvidos que serão descritas detalhadamente na secção seguinte.