INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA MECÂNICA




Aplicação de Machine Learning na Parametrização de Meta-Heurísticas para resolver Problemas de Escalonamento das Operações

DANIEL JOAQUIM DA SILVA DIAS
Setembro de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Aplicação de Machine Learning na Parametrização de Meta-Heurísticas para Resolver Problemas de Escalonamento das Operações

Daniel Joaquim da Silva Dias

Dissertação para obtenção do Grau de Mestre em Engenharia Mecânica, Área de Especialização em Gestão Industrial

Orientador: André Borges Guimarães Serra e Santos

Co-orientador: Maria Leonilde Rocha Varela

Porto, setembro de 2024

Agradecimentos

A realização deste trabalho representa a culminação de uma etapa importante da minha vida académica e profissional, sendo o resultado de muitos anos de dedicação, esforço e, acima de tudo, do apoio e colaboração de muitas pessoas a quem devo o meu profundo agradecimento.

Em primeiro lugar, gostaria de expressar a minha sincera gratidão ao Instituto Superior de Engenharia do Porto (ISEP), instituição que me proporcionou a formação e os recursos necessários para o desenvolvimento deste trabalho. Ao longo deste percurso, tive o privilégio de me relacionar e aprender com professores que deixaram uma marca significativa na minha trajetória, e por isso agradeço-lhes pela dedicação e pelo apoio contínuo.

Quero expressar um agradecimento, de forma especial, ao meu orientador, Engenheiro André Serra e Santos, pela orientação constante, pela partilha do seu vasto conhecimento e pela confiança depositada em mim ao longo deste percurso. O seu suporte e aconselhamento foram fundamentais para o sucesso deste projeto.

Um agradecimento igualmente à minha co-orientadora, Engenheira Leonilde Varela, da Universidade do Minho, pela sua valiosa contribuição, pelo apoio constante e pela visão estratégica que trouxe uma perspetiva enriquecedora ao trabalho.

Aos meus pais e à minha irmã, dedico uma palavra de profundo reconhecimento. Sem o vosso apoio incondicional e compreensão, nada disto teria sido possível. Vocês estiveram presentes em todos os momentos do meu percurso académico, foram o meu suporte durante a realização deste trabalho, e por isso estou eternamente grato.

Aos meus amigos, que partilharam comigo esta jornada, agradeço por todo o suporte, pelas palavras de incentivo e por todos os momentos de descontração. A vossa amizade foi e continua a ser um pilar fundamental na minha vida.

Por fim, a todos aqueles que, de uma forma ou de outra, contribuíram para a realização deste trabalho, deixo aqui um sincero agradecimento. Cada um de vocês desempenhou um papel essencial na concretização deste projeto, e este sucesso é, em grande parte, também vosso.

Resumo

Os Problemas de Escalonamento são desafios comuns em diversas áreas, e envolvem a procura por soluções ótimas em espaços de soluções que, muitas vezes, crescem exponencialmente com a dimensão do problema. Enumerar todas as soluções torna-se rapidamente inviável. Como alternativa, surgem as Meta-Heurísticas, que são eficazes, oferecendo estratégias flexíveis e robustas para encontrar soluções boas em intervalos de tempo reduzidos. No entanto, o desempenho destas técnicas depende fortemente da escolha adequada dos valores dos seus parâmetros. Neste trabalho, propõe-se um Protótipo de um *Framework* de Parametrização de Meta-Heurísticas, no qual um Agente Inteligente é responsável pela seleção e ajuste dinâmico dos valores dos parâmetros do *Simulated Annealing*, que é a técnica implementada para resolver o Problema de Escalonamento de Minimização do *Makespan* em *Job-Shop*. O Agente utiliza o algoritmo *Q-Learning* para guiar o processo de parametrização. À medida que recebe recompensas com base no desempenho de cada configuração, o Agente ajusta os valores dos parâmetros e refina a sua política de decisão de forma contínua, aprendendo assim a selecionar os parâmetros mais adequados. Como consequência, este processo progressivo permite que a Meta-Heurística se torne cada vez mais eficiente na procura pelas melhores soluções. Para avaliar o desempenho do modelo, foi conduzido um estudo computacional focado na eficiência do processo de aprendizagem do Agente e na eficácia das diferentes combinações de parâmetros que permitem obter a melhor solução. O Protótipo encontrou a melhor solução em 85% das instâncias, e, nas restantes, as melhores soluções aproximam-se significativamente das melhores documentadas na literatura, com diferenças que não são expressivas. Em instâncias de menor dimensão, o Agente não só encontrou a melhor solução, como também adquiriu a capacidade de fazê-lo num número reduzido de iterações. Por sua vez, em espaços de soluções mais irregulares, o sistema obteve a melhor solução, identificou as combinações de parâmetros que permitiram alcançá-la, tendeu a revisitá-las, no entanto, em vários momentos, a mesma configuração de parâmetros nem sempre produziu os mesmos resultados consistentemente, o que, em alguns casos, dificultou o processo de aprendizagem do Agente e afetou a qualidade dos resultados. Mesmo assim, o Protótipo demostrou ser uma excelente estratégia para ajustar, de forma contínua e inteligente, os parâmetros de uma Meta-Heurística.

Palavras-chave: Problema de Escalonamento, Meta-Heurísticas, Parametrização, Aprendizagem por Reforço, *Q-Learning*, *Simulated Annealing*

Abstract

Scheduling Problems are common challenges across various fields and involving the search for optimal solutions within a solution spaces that, often, grow exponentially with the problem's size. Enumerating all possible solutions quickly becomes infeasible. As an alternative, Meta-Heuristics have emerged as effective approaches, offering flexible and robust strategies for finding good solutions requiring low computational times. However, the performance of these techniques is highly dependent on the appropriate selection of their parameter values. In this work, a Prototype of a Meta-Heuristic Parameterization Framework is proposed, wherein an Intelligent Agent is responsible for the dynamic selection and adjustment of the parameters of Simulated Annealing, which is the technique implemented to solve the Job-Shop Scheduling Problem, specifically for minimizing makespan. The Agent employs the Q-Learning algorithm to guide the Meta-Heuristic parameter tuning process. As it receives rewards based on the performance of each parameter configuration, the Agent continuously adjusts the parameters values and refines its decision-making policy, progressively learning to select the most suitable parameters. As a results, this process enables the Meta-Heuristic to become increasingly efficient in the search for optimal solutions. To evaluate the model's performance, a computational study was conducted, focusing on the efficiency of the Agent's learning process and the effectiveness of different parameter configurations that lead to optimal solutions. The Prototype identified the optimal solution in 85% of the instances, and in the remaining cases, the solutions obtained were significantly close to the best documented in the literature, with negligible differences. In instances of smaller dimensions, the Agent not only found the best solution but also acquired the ability to achieve it within a reduced number of iterations. Conversely, when the solution space is more irregular, the system successfully achieved the best solution and identified the parameter configurations that yield the best result. It tended to revisit these combinations; however, in several instances, the same parameter configuration did not consistently produce the same results, which, in some cases, hindered the Agent's learning process and affected the quality of the results. Nevertheless, the Prototype proved to be an excellent strategy for continuously and intelligently tuning the parameters of a Meta-Heuristic.

KEYWORDS: Scheduling Problem, Meta-Heuristics, Parameter Tuning, Reinforcement Learning, Q-Learning, Simulated Annealing

Índice

Lista de Figuras xii

Lista de Tabelas xv

Acrónimos e Siglas xvii

1. Introdução 1
   1.1. Contextualização 1
   1.2. Objetivos 2
   1.3. Metodologia 3
   1.4. Estrutura do Documento 4
2. Revisão Bibliográfica 5
   2.1. Escalonamento das Operações 5
      2.1.1. Problema de Afetação 7
      2.1.2. Problema de Calendarização e Sequenciação 7
   2.2. Definições e Noções Básicas 8
   2.3. Representação de um Problema de Escalonamento 9
      2.3.1. Ambiente de Produção 10
      2.3.2. Restrições de Processamento 11
      2.3.3. Critérios de Otimização 12
   2.4. Métodos de Otimização 14
      2.4.1. Métodos Exatos 14
      2.4.2. Métodos Aproximados 17
      2.4.3. Conceitos Básicos 19
      2.4.4. Pesquisa Local 22
   2.5. Meta-Heurísticas 23
      2.5.1. Simulated Annealing 27
      2.5.2. Tabu Search 30
      2.5.3. Outras Meta-Heurísticas 32
   2.6. Parametrização de Meta-Heurísticas 34
      2.6.1. Parametrização *Offline* 35
      2.6.2. Parametrização *Online* 38
      2.6.3. Parametrização do *Simulated Annealing* 39
   2.7. Machine Learning 42
   2.8. Tipos de Aprendizagem 43
      2.8.1. Aprendizagem Supervisionada 43
      2.8.2. Aprendizagem Não Supervisionada 44
      2.8.3. Aprendizagem por Reforço *(Reinforcement Learning)* 45
   2.9. Aplicação de ML em Meta-Heurísticas 52

2.10. Análise Bibliométrica 58
2.10.1. Revisão do Estado da Arte - Estudo de Casos 62
2.10.2. Discussão e Caminhos Futuros 68

3. Protótipo de Parametrização de Meta-Heurísticas 71
3.1. Problema de Escalonamento 71
3.2. *Framework* de Parametrização através de RL 75
3.2.1. Leitor de Instâncias 77
3.2.2. Sistema Inteligente de Parametrização 78
3.3. Protótipo do *Framework* de Parametrização 82
3.3.1. *Simulated Annealing* - Formalização do Ambiente e das Ações 83
3.3.2. Agente de Parametrização – Treino e Parâmetros do algoritmo de RL 86

4. Estudo Computacional, Resultados e Discussão 93
4.1. Estudo Computacional 93
4.2. Resultados do Protótipo do Framework de Parametrização em Problemas de Minimização do *Makespan* em *Job-Shop* 98
4.2.1. Qualidade das Soluções 98
4.2.2. Evolução do Melhor *Makespan* e do Número de Iterações por Episódio 99
4.2.3. Qualidade da Configuração Ótima de Parâmetros 108
4.3. Discussão de Resultados 114

5. Conclusão 117
5.1. Conclusões Finais 117
5.2. Limitações e Trabalhos Futuros 120

Referências 123

Declaração de Integridade 143

Apêndice A 145

Apêndice B 149

Apêndice C 157

Lista de Figuras

Figura 1- Representação de um Problema de Afetação (adaptado de [12]) 7
Figura 2- Representação de um Problema de Sequenciação (adaptado de [12]) 8
Figura 3- Métodos de otimização [26] 14
Figura 4- Árvore de pesquisa desenvolvida na aplicação do método *Branch and Bound* [29].. 15
Figura 5- Tipos de Codificação para Problemas de Otimização 20
Figura 6- Estruturas de Vizinhança comuns em vários tipos de problemas 22
Figura 7- Intensificação, Diversificação e Ótimos Locais e Globais (adaptado de [70]).... 25
Figura 8- Estratégia de funcionamento do SA para evitar ótimos locais (adaptado de [26]).... 28
Figura 9- Tipos de Parametrização (adaptado de [12, 26, 117]).... 35
Figura 10- Classificação versus Regressão na Aprendizagem Supervisionada.... 44
Figura 11- Relação entre Agente e Ambiente (adaptado de [189]).... 46
Figura 12- Representação do problema *"GridWorld"*.... 51
Figura 13- Resultados alcançados pelo Agente Aleatório no problema *"GridWorld"* 52
Figura 14- Resultados alcançados pelo Agente *Q-Learning* no problema *"GridWorld"* 52
Figura 15- Representação de um Problema de Seleção do Algoritmo (adaptado de [220]) 53
Figura 16- Distribuição temporal das publicações entre 1998 e julho de 2024 60
Figura 17- As dez principais áreas de investigação das publicações.... 61
Figura 18- Mapa de Coocorrência de Palavras-chave.... 62
Figura 19- Grafo disjuntivo para a instância de um problema 3x3 *Job Shop*.... 75
Figura 20- Diagrama representativo do *Framework* de Parametrização através de RL 76
Figura 21- Exemplo de uma instância do problema *Job-Shop*: Formato Standard [7] 78
Figura 22- Fluxo de informação entre os componentes do AIP.... 78
Figura 23- Estrutura de cada módulo que constitui o SA.... 82
Figura 24- Exemplo da representação de uma solução 86
Figura 25- Diagrama representativo do processo de treino do Agente 90
Figura 26- Resultados da execução da Meta-Heurística para a instância la02 com 3500 iterações como Critério de Interrupção do SA 96
Figura 27- Evolução da melhor solução por episódio: instância la01 100
Figura 28- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la01.... 101
Figura 29- Evolução da melhor solução por episódio: instância la04 101
Figura 30- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la04.... 102
Figura 31- Evolução da melhor solução por episódio: instância la06 103
Figura 32- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la06.... 104
Figura 33- Evolução da melhor solução por episódio: instância la12 104
Figura 34- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la12.... 105
Figura 35- Evolução da melhor solução por episódio: instância la18 105

Figura 36- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la18 106
Figura 37- Evolução da melhor solução por episódio: instância la20 107
Figura 38- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la20 107
Figura 39- Distribuição temporal das publicações entre 1997 e dezembro de 2024 150
Figura 40- As dez principais áreas de investigação das publicações 151
Figura 41- Mapa de Coocorrência de Palavras-chave nos artigos sobre *Machine Learning* e Escalonamento das Operações 151

Lista de Tabelas

Tabela 1- Regras de Prioridade (Adaptado de [13, 18]) 19
Tabela 2- Algoritmo genérico representativo da Pesquisa Local (adaptado de [26, 58]) 23
Tabela 3- Diferentes formas de classificação das Meta-Heurísticas 25
Tabela 4- Algoritmo representativo do *Simulated Annealing* (adaptado de [26]) 27
Tabela 5- Algoritmo representativo do *Tabu Search* (adaptado de [26, 93]) 32
Tabela 6- Algoritmo representativo do *Q-Learning* [208] 49
Tabela 7- Pesquisa por Palavras-Chave 60
Tabela 8- Classificação de 18 artigos relevantes identificados 63
Tabela 9- Exemplo de uma instância de um problema 3x3 *Job Shop* (n=3, m=3) [284] 75
Tabela 10- Valores dos parâmetros para o algoritmo de RL 88
Tabela 11- Intervalos dos parâmetros que caracterizam o SA nas instâncias la01 a la20 94
Tabela 12- Valores dos parâmetros que caracterizam o *Q-Learning* e o processo de treino 95
Tabela 13- Comparação dos resultados obtidos pelo Protótipo com os valores da literatura nas instâncias la01 a la20 99
Tabela 14- Frequências das combinações de parâmetros que conduzem à melhor solução nas instâncias la01 a la20 108
Tabela 15- Resultados da execução da MH com cada configuração ótima de parâmetros nas instâncias la01 a la20 112
Tabela 16- Pesquisa por Palavras-Chave 150
Tabela 17- Outros artigos relevantes sobre as duas áreas importantes em análise 154

Acrónimos e Siglas

Lista de Acrónimos e Siglas

ACO	*Ant Colony Optimization*
ANN	*Artificial Neural Network*
AO	Ambiente de Otimização
AP	Agente de Parametrização
BN	*Bayesian Network*
BRKGA	*Biased Random-Key Genetic Algorithm*
BRKGA-QL	*Biased Random-Key Genetic Algorithm Q-Learning*
CBR	*Case-Based Reasoning*
CLONALG	*Clonal Selection Algorithm*
CPU	*Central Processing Unit*
CR	*Cooling Rate*
DASA	*Differential Ant-Stigmergy Algorithm*
DDPG	*Deep Deterministic Policy Gradient*
DE	*Differential Evolution*
DEMOCA	*Deep Reinforcement Learning Multi-Objective Control Algorithm*
DOE	*Design of Experiments*
DRL	*Deep Reinforcement Learning*
DSR	*Design Science Research*
DTM	*Deterministic Turing Machine*
EA	*Evolutionary Algorithm*
EL	*Epoch Length*
EV	Estrutura de Vizinhança
FR	Fator de Recompensa
FSS	*Fish School Search*
GA	*Genetic Algorithm*
HCLPSO	*Heterogeneous Comprehensive Learning Particle Swarm Optimization*
IA	Inteligência Artificial
IDA^*	*Iterative-Deepening* A^*
ISEP	Instituto Superior de Engenharia do Porto
JSSP	*Job-Shop Scheduling Problem*

LS	*Local Search*
MH	Meta-Heurística
ML	*Machine Learning*
NDTM	*Non-Deterministic Turing Machine*
NMMS	Número de Melhorias na Melhor Solução
NP	*Non-Deterministic Polynomial Time*
PPO	*Proximal Policy Optimization*
PSO	*Particle Swarm Optimization*
REGFOR	*Random Regression Forests*
RL	*Reinforcement Learning*
RPD	*Relative Percentage Deviation*
REVAC	*Relevance Estimation and Value Calibration*
SA	*Simulated Annealing*
SIP	Sistema Inteligente de Parametrização
SMAC	*Sequential Model-based Algorithm Configuration*
SPO	*Sequential Parameter Optimization*
SVM	*Support Vector Machine*
TI	Temperatura Inicial
TS	*Tabu Search*
TSP	*Traveling Salesman Problem*

1. Introdução

A Introdução visa explorar, de forma abrangente, o que motiva a realização deste trabalho. Inicialmente, é apresentado o ambiente no qual a dissertação está inserida, e justifica-se a relevância da mesma. De seguida, são apresentados os objetivos e a metodologia de investigação, que descreve o conjunto de procedimentos, técnicas e ferramentas utlizados para alcançar os objetivos propostos. A secção termina com a organização do documento.

1.1. Contextualização

Num cenário empresarial cada vez mais dinâmico e competitivo, as organizações enfrentam constantes desafios para se destacar e atender à procura crescente do mercado. A competitividade tornou-se um fator determinante para a sobrevivência e sucesso dos negócios, exigindo das empresas uma capacidade contínua de adaptação para serem capazes de oferecer produtos ou serviços de alta qualidade, com preços competitivos e prazos de entrega reduzidos. Neste contexto, emerge a importância do Escalonamento das Operações como uma estratégia essencial para satisfazer a procura de forma eficiente.

O Escalonamento das Operações envolve a alocação dos recursos às operações e a sua distribuição no tempo. Quando realizado de maneira adequada, é capaz de impactar positivamente o desempenho e a eficiência de uma organização através da otimização de recursos, da redução de custos operacionais e da diminuição dos tempos de resposta aos pedidos dos clientes [1]. Além disso, é um processo que envolve problemas complexos porque o número de soluções de um problema cresce exponencialmente em função da dimensão do problema. Para abordar os problemas desta área, existem métodos aproximados que exploram exaustivamente o espaço de soluções. Considerando que, em muitos problemas, torna-se complicado (e até mesmo impossível) enumerar as soluções, pode-se recorrer a técnicas aproximadas que são capazes de encontrar soluções boas em intervalos de tempo razoáveis [2].

No campo das técnicas aproximadas, destacam-se as Meta-Heurísticas, que são estratégias de otimização que utilizam princípios de pesquisa global e técnicas adaptativas para explorar eficientemente o espaço de soluções. De maneira geral, a identificação do valor de cada parâmetro que caracteriza cada técnica, e etapas do processo de procura por soluções como gerar a solução inicial, gerar a vizinhança e avaliar as respetivas soluções, selecionar o movimento a realizar, equilibrar as estratégias de intensificação e diversificação, entre outras, assumem uma importância crucial na resolução de Problemas de Otimização, o que faz com que a pesquisa por soluções capazes de as otimizar seja um desafio contínuo [3].

O desempenho de uma Meta-Heurística é significativamente influenciado pelos valores dos parâmetros que a caracterizam. A parametrização destes algoritmos é um desafio significativo porque envolve a identificação de combinações de parâmetros que proporcionem um equilíbrio adequado entre a exploração e a intensificação do espaço de soluções. Embora existam diversas técnicas para parametrizar uma Meta-Heurística, algumas ainda requerem uma quantidade significativa de experiências ou exigem um utilizador com um elevado nível de experiência para alcançar bons resultados. A necessidade de experimentar manualmente múltiplas configurações de parâmetros pode ser demorada e impraticável, limitando a eficiência e a eficácia da abordagem mais tradicional.

Considerando o panorama da Inteligência Artificial (IA) nos últimos anos, o *Machine Learning (ML)* surge como uma área emergente capaz de oferecer soluções inovadores em diversos domínios. No contexto das Meta-Heurísticas, a utilização de técnicas de ML oferece uma oportunidade notável para automatizar o processo de ajuste de parâmetros, permitindo assim não apenas reduzir significativamente o tempo necessário para identificar as melhores configurações de parâmetros, mas também adaptar dinamicamente os parâmetros em resposta ao comportamento do algoritmo durante a execução, o que resulta numa exploração mais eficiente do espaço de soluções e na obtenção de resultados de melhor qualidade.

1.2. Objetivos

Neste trabalho, pretende-se superar os desafios que as abordagens atuais de resolução de Problemas de Escalonamento enfrentam, através da integração de algoritmos de ML com Meta-Heurísticas. Considerando que o ML tem apresentado um crescimento significativo nos últimos tempos, o objetivo é explorar a capacidade deste campo de IA fornecer *insights* e orientações para melhorar o processo de aplicação das técnicas de otimização tradicionais na resolução de Problemas de Escalonamento.

A estrutura proposta utiliza Aprendizagem por Reforço (RL, de *Reinforcement Learning*) para parametrizar Meta-Heurísticas de forma automática e dinâmica. Este processo envolve a utilização de algoritmos de RL para treinar um Agente capaz de selecionar as configurações de parâmetros que maximizem a qualidade das soluções encontradas pela Meta-Heurística. Num processo de constante interação entre o Agente e um Ambiente de Escalonamento, no qual se insere o problema a resolver e a Meta-Heurística implementada, o mecanismo de feedback visa fornecer recompensas adequadas ao Agente, incentivando-o a explorar as configurações de parâmetros mais eficazes.

Assim, podem ser identificadas as principais contribuições desta dissertação:

- → Revisão do estado da arte das abordagens de resolução de Problemas de Otimização Combinatória. Análise das diversas técnicas para perceber os parâmetros que as constituem e as características que podem ser exploradas;
- → Estudo sobre os algoritmos de ML aplicados na parametrização de Meta-Heurísticas ao longo dos últimos anos. Análise aos trabalhos disponíveis da literatura através de uma análise bibliométrica e da identificação dos estudos mais relevantes.

→ Especificação da arquitetura do *Framework* desenvolvido para parametrizar Meta-Heurísticas através da utilização de modelos de ML, o que inclui a descrição completa dos princípios que o caracterizam;

→ Desenvolvimento e implementação de um Protótipo de Parametrização, caracterizado pela implementação de algoritmos de RL e do *Simulated Annealing (SA)*. Descrição completa da definição do espaço de estados e ações, do mecanismo de recompensa, da estratégia adotada para minimizar o impacto da natureza estocástica das Meta-Heurística, e do processo de treino;

→ Realização de um estudo computacional para comparação do trabalho desenvolvido com as estruturas propostas em outros sistemas. Análise estatística dos resultados para identificar as vantagens associadas a cada solução e os respetivos contributos.

1.3. Metodologia

Definir o modo como a pesquisa e o trabalho devem ser realizadas é importante para que os objetivos desta dissertação sejam alcançados. Neste trabalho, a conceção da metodologia de investigação segue a estrutura "*Research Onion*", proposta por Saunders et al. [4]. A filosofia de investigação é o pragmatismo e segue uma abordagem dedutiva, pois formula-se uma estratégia de investigação para testar uma teoria desenvolvida. É utilizado um multi-método quantitativo centrado no *Design Science Research (DSR)*, e o horizonte temporal é transversal.

O DSR é uma metodologia proposta por Hener et al. [5] para sistemas de informação, que enfatiza o desenvolvimento sistemático e a avaliação de artefactos destinados a resolver problemas. É importante realçar três atividades gerais [6]:

1. Construção de um artefacto baseado em conhecimentos adquiridos na prática ou na análise da teoria;
2. Recolha de dados sobre o desempenho funcional do artefacto, ou seja, avaliação da sua qualidade, utilidade e eficácia;
3. Reflexão sobre o processo de construção e as implicações que os dados recolhidos têm para o artefacto, o que engloba a perceção da contribuição dos resultados obtidos.

A primeira atividade corresponde ao desenvolvimento de uma estrutura que usa algoritmos de ML para parametrizar Meta-Heurísticas. A segunda e a terceira envolvem a recolha, o tratamento e a subsequente análise crítica dos dados. Neste contexto, serão utilizadas instâncias de Problemas de Otimização disponíveis na OR-Library, uma das bases de dados mais reconhecidas dentro da área [7]. Este procedimento possibilitará a comparação dos resultados obtidos com os apresentados por outros autores, considerando critérios como a melhor solução identificada.

Para a Revisão Bibliográfica, recorre-se ao método *Snowballing*, No primeiro estudo sobre os dois principais domínios deste trabalho, Escalonamento das Operações e *Machine Learning*, onde o objetivo foi perceber os conceitos fundamentais de cada área, selecionaram-se artigos relevantes. Com a análise das referências destes artigos, foi possível obter outros estudos de

qualidade, capazes de fornecer uma visão mais aprofundada e completar a informação recolhida anteriormente, formando assim uma cadeia sequencial de pesquisa [8]. A interseção entre os dois temas compreendeu a análise bibliométrica e a investigação de casos de estudo pertinentes, resultantes da definição de uma *query* que permitiu obter as publicações mais relevantes para o tema deste trabalho.

1.4. Estrutura do Documento

O presente documento está organizado em cinco capítulos.

O primeiro capítulo, intitulado "Introdução", inicia-se com uma breve contextualização, seguida das apresentações dos objetivos, da metodologia adotada e da estrutura do relatório.

No segundo capítulo, "Revisão Bibliográfica", são contextualizadas as duas áreas fundamentais deste estudo: Escalonamento das Operações e *Machine Learning*. Este capítulo explora os conceitos essenciais relacionados com o Problema do Escalonamento e enfatiza a importância do uso de métodos aproximados, especificamente Meta-Heurísticas, na resolução de Problemas de Otimização. São apresentados os princípios que caracterizam algumas Meta-Heurísticas, como o *Simulated Annealing* e o *Tabu Search* (TS). Subsequentemente, são introduzidas as noções básicas sobre ML, com uma ênfase especial nos tipos de aprendizagem e nos algoritmos associados. O capítulo conclui com uma exploração da inter-relação entre as duas áreas principais através de uma análise bibliométrica e de estudo de casos focados na aplicação de técnicas de ML na parametrização de Meta-Heurísticas.

O terceiro capítulo, "Protótipo de Parametrização de Meta-Heurísticas", descreve o desenvolvimento de um *Framework* de Parametrização de Meta-Heurísticas para a resolução de Problemas de Escalonamento em *Job-Shop*, bem como a respetiva implementação de um Protótipo correspondente. Neste contexto, são mencionados o algoritmo de RL e a Meta-Heurística implementados, bem como o processo de formalização de um problema que integre RL, o que abrange a definição dos espaços de estados e ações e do mecanismo de recompensa.

O quarto capítulo, "Estudo Computacional, Resultados e Discussão", é dedicado ao estudo computacional realizado para análise e validação dos resultados obtidos pelo Protótipo do *Framework* de Parametrização, bem como à análise e discussão dos resultados. Inicialmente, são introduzidos os parâmetros que caracterizam o processo de treino/resolução para cada instância, seguida da análise do processo de aprendizagem do Agente e da qualidade das soluções obtidas e das diferentes combinações de parâmetros.

Por fim, as conclusões são apresentadas no quinto capítulo, intitulado "Conclusão", com destaque para a identificação e enumeração das contribuições deste trabalho, das limitações enfrentadas durante a sua execução, e das sugestões de trabalhos futuros que podem constituir excelentes estratégias para melhorar e estender a aplicabilidade do que é apresentado neste documento.

2. Revisão Bibliográfica

A Revisão Bibliográfica é dedicada ao enquadramento técnico e científico de duas áreas cruciais nesta dissertação - Escalonamento das Operações e *Machine Learning*. Serão apresentados os conceitos fundamentais relacionados com o Escalonamento, as formas mais comuns de representar este tipo de problema, bem como os dois principais tipos de métodos de otimização - exatos e aproximados, que refletem o modo como as soluções podem ser obtidas. Além disso, será discutida a importância da parametrização nesses métodos, nomeadamente em Meta-Heurísticas, com ênfase nas diferentes técnicas que podem ser aplicadas.

Relativamente ao ML, serão expostos os diferentes tipos de aprendizagem, bem como os principais algoritmos que os constituem. O Capítulo 2 terminará com uma análise à interseção entre as duas temáticas principais, com destaque para o modo como se combinam para oferecer soluções inteligentes e eficazes.

2.1. Escalonamento das Operações

O Escalonamento das Operações é crucial porque distribui a capacidade de uma organização pelas necessidades do mercado. É um processo de tomada de decisão que lida com a alocação de recursos a operações. Dependendo da situação, os recursos podem assumir diferentes formas, como, por exemplo, equipamentos de produção, pistas de um aeroporto, etc. Por sua vez, as atividades, que são aquilo que necessita de ser realizado pelos recursos, podem ser, por exemplo, tarefas num processo de fabrico ou descolagens num aeroporto [9].

Em meados do século XX, o Escalonamento das Operações emergiu como um tópico de considerável interesse entre a comunidade de investigação, o que culminou com a sua categorização como sendo um Problema de Otimização, e conduziu à publicação dos primeiros métodos de resolução [9, 10]. É uma área que abrange uma variedade de atividades numa organização, e não se limita apenas à produção. Problemas deste tipo manifestam-se em diversos contextos de instalações de produção industrial, abrangendo setores como a indústria de papel e celulose, petrolífera e gás, militar, química, alimentar, entre outras, onde é necessário processar determinadas atividades ou operações em recursos específicos [11].

O Escalonamento das Operações é realizado durante a Programação da Produção, ou seja, na última fase do Planeamento e Controlo da Produção, o que faz com que esteja associado a um elevado nível de detalhe e a um horizonte de curto prazo [12].

Pinedo, em [13], classifica o processo de Escalonamento como uma função fundamental do Planeamento e Programação da Produção, frequentemente utilizada pelas empresas nos seus processos. Trata-se de alocar determinados recursos a um conjunto de operações em períodos específicos e determinar o calendário de atividades que otimiza uma ou mais medidas de desempenho.

Brucker, em [14], menciona que o Escalonamento de um sistema de produção se caracteriza pela procura e seleção de uma programação que satisfaça determinadas restrições. É um processo que depende das características únicas de cada ambiente de produção, bem como das ordens de fabrico existentes, dos recursos e das máquinas disponíveis, e das operações que podem ser executadas em cada uma delas.

Normalmente, a identificação e a modelação das características de um ambiente de produção determinam a natureza do tipo de Escalonamento. É importante mencionar as seguintes:

- → **Escalonamento Determinístico**, que assume que os parâmetros dos problemas são conhecidos à priori, o que é comum em problemas académicos [13];
- → **Escalonamento Estocástico**, baseado no princípio de que não existe um conhecimento prévio das condições dos problemas, considerando assim a incerteza e a aleatoriedade associada aos dados de entrada. Normalmente, são utilizadas distribuições estatísticas para modelar os dados de um problema, e apenas nos momentos em que as variáveis ocorrem é que são conhecidos os seus valores concretos [13];
- → **Escalonamento Estático**, que considera que o plano de atividades definido na fase inicial permanece inalterado, mesmo que novos trabalhos surjam no decorrer do processo. Os programas criados com base neste princípio podem-se tornar obsoletos após libertados para o chão de fábrica [15];
- → **Escalonamento Dinâmico**, caracterizado por considerar os acontecimentos em tempo real, por exemplo, a chegada de atividades urgentes, avarias nas máquinas e alterações das datas de entrega [15, 16]. O plano de escalonamento é alterado à medida que novas atividades são adicionadas ao sistema [17].

Para definir um problema deste tipo é necessário conhecer aprofundadamente não apenas o conjunto de operações a executar, mas também os recursos necessários para as realizar. Neste sentido, Baker & Trietsch [18] abordam o Escalonamento de Operações como sendo um processo de tomada de decisão que responde a duas questões:

- → Como devem ser distribuídas as operações pelos recursos?
- → Como devem ser distribuídas as operações no tempo?

Estas duas questões evidenciam a ideia clara de que o Escalonamento é um processo que envolve a atribuição de operações a recursos ou vice-versa (decisões de afetação), a definição da ordem temporal de processamento (decisões de sequenciação), e a definição dos tempos de processamento inicial e final de cada atividade num recurso (decisões de calendarização) [19].

2.1.1. Problema de Afetação

Um Problema de Afetação (ou Alocação) verifica-se quando há várias operações para processar, e existem múltiplos recursos capazes de o fazer. O objetivo é distribuir as operações pelos recursos, ou vice-versa [20].

Este tipo de decisões é frequente em vários contextos, não se limitando apenas à produção. Numa equipa de projeto, a atribuição das responsabilidades de uma atividade a um conjunto de recursos humanos é um exemplo de uma decisão de alocação de equipas.

Uma representação do Problema de Afetação pode ser analisada na Figura 1.

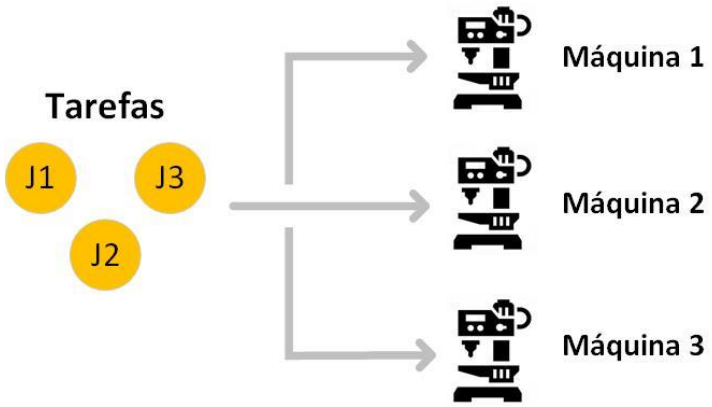


Figura 1- Representação de um Problema de Afetação (adaptado de [12])

Para Blazewicz et al. [20], o problema de afetação é descrito da seguinte forma: "Conhecidas *n* operações pertencentes a $T = \{T_1, T_2, \dots, T_n\}$, *m* máquinas pertencentes a $P = \{P_1, P_2, \dots, P_m\}$, e *s* recursos adicionais pertencentes a $R = \{R_1, R_2, \dots, R_s\}$, como deverão as máquinas de *P* e recursos de *R*, ser afetados às operações de *T* de forma a maximizar/minimizar uma determinada medida de desempenho?". Por outras palavras, o intuito é distribuir as operações pelas máquinas e recursos, considerando sempre as restrições do ambiente em questão.

2.1.2. Problema de Calendarização e Sequenciação

Um Problema de Calendarização ou Sequenciação compreende decisões que sucedem a afetação, e corresponde à distribuição de um conjunto de operações no tempo do processamento [18].

Em [21], Santos aborda a necessidade de separar os conceitos de calendarização e sequenciação porque, embora possuam aspetos em comum, não têm sempre o mesmo significado. Os Problemas de Sequenciação determinam a sequência de execução das operações, independentemente de os tempos de execução serem difusos. Por sua vez, os Problemas de Calendarização ocorrem depois da sequenciação e determinam quando é que as operações devem ser processadas.

Numa situação onde se possuem *n* operações, $T = \{T_1, T_2, \dots, T_n\}$, e uma sequência de fabrico num recurso, S= $\{S_1, S_2, \dots, S_n\}$, uma decisão de sequenciação refere-se à ordenação do processamento de cada operação de T na sequência S do respetivo recurso com o objetivo de

otimizar uma ou mais medidas de desempenho. Por sua vez, quando existem *n* operações, $T = \{T_1, T_2, \dots, T_n\}$, m máquinas pertencentes a $P = \{P_1, P_2, \dots, P_m\}$, uma sequência de fabrico, $S = \{S_1, S_2, \dots, S_n\}$, para cada máquina, e um intervalo de tempo $[a, b]$, distribuir as operações de *T* no intervalo de tempo para otimizar uma medida de desempenho é representativo de uma decisão de calendarização [12]. Este último tipo de problema, calendarização, estabelece assim a hora de início de execução de cada operação.

Uma representação do Problema de Sequenciação pode ser analisada na Figura 2.

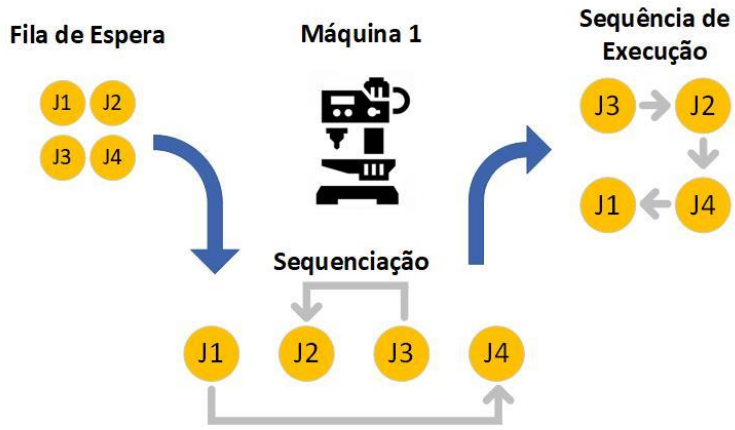


Figura 2- Representação de um Problema de Sequenciação (adaptado de [12])

Nos Problemas de Escalonamento em que as características dos problemas são conhecidas sem incerteza, ou seja, em problemas determinísticos, a sequenciação e a calendarização podem assumir-se como semelhantes, o que pode ter como consequência uma programação ótima sem atrasos, uma vez que não é possível representar intervalos de espera entre duas atividades [12, 13]. Isto mostra que, de forma geral, a calendarização possibilita a obtenção de um plano mais detalhado, capaz de incluir informação mais relevante para o problema.

2.2. Definições e Noções Básicas

Como ponto de partida para abordar um Problema de Escalonamento, é crucial descrever os conceitos base e fundamentais – atividade ou tarefa, operação, máquina ou recurso, e oficina.

Uma *Atividade ou Tarefa* é uma unidade de trabalho que deve ser realizada num determinado período. Uma *Operação* é uma etapa específica ou uma parte indivisível de uma atividade a ser realizada num único recurso. Por sua vez, um *Recurso ou Máquina* é uma unidade de processamento capaz de executar uma ou mais operações, e possui as suas próprias características que influenciam esse processo. A *Oficina* refere-se ao local onde os recursos estão disponíveis para serem utilizados no processamento das diversas atividades [12, 18].

No que diz respeito à estrutura de representação, é habitual representar o número de atividades por *n*, o número de máquinas por *m*. Tendo em conta que *i* representa as máquinas e *j* representa as atividades, os vários elementos podem ser definidos como [13]:

→ $\boldsymbol{p_{ij}}$ **-** Expressa o tempo de processamento da atividade *j* na máquina *i*. Quando uma atividade é executada exclusivamente numa máquina, ou o tempo de processamento da mesma é independente da máquina onde é executada, representa-se por p_j;

→ $\boldsymbol{r_j}$ – Expressa a data de lançamento da atividade *j*, ou seja, o momento a partir do qual a mesma pode ser processada;

→ $\boldsymbol{d_j}$ – Refere-se à data em que a atividade *j* deve ser entregue ao cliente. Em situações onde este prazo não é cumprido, o cliente pode aceitar receber a encomenda, mas esta ser associada a uma penalização;

→ $\boldsymbol{w_j}$ - Representa o peso da atividade *j* no sistema produtivo da empresa, isto é, a importância da mesma em relação às outras atividades do sistema;

→ $\boldsymbol{C_j}$ **–** É o tempo de conclusão da atividade *j* na última máquina da sua sequência;

→ $\boldsymbol{E_j}$ - Representa a conclusão antecipada de uma atividade em relação à sua data definida. A Equação (1) apresenta o cálculo deste parâmetro, que é definido como a diferença entre a data de entrega e o tempo de conclusão de uma atividade.

$$E_j = máx(d_j - C_j\ , 0) \tag{1}$$

→ $\boldsymbol{L_j}$ **-** Representa o atraso ou antecipação de uma atividade em relação à data delineada. A Equação (2) apresenta o cálculo deste parâmetro, que resulta da diferença entre o tempo de conclusão e a data de entrega de uma atividade;

$$L_j = C_j - d_j \tag{2}$$

→ $\boldsymbol{T_j}$ **–** Expressa o atraso de uma atividade em relação à data definida. É semelhante a L_j, no entanto nunca apresenta um valor negativo. A Equação (3) apresenta o seu cálculo.

$$T_j = máx(C_j - d_j\ , 0) \tag{3}$$

2.3. Representação de um Problema de Escalonamento

Cada empresa possui as suas próprias características, a sua forma específica de operar e os seus próprios recursos nas quantidades adequadas para satisfazer as suas necessidades, o que faz com que os Problemas de Escalonamento possuam habitualmente características consideravelmente distintas.

Tendo em conta a importância desta área em análise e o elevado número de parâmetros a considerar nos diversos tipos de problemas, foram desenvolvidas, ao longo do tempo, várias nomenclaturas e formas de representação, tal como Varela analisa em [22]. Conway et al. [23], um dos primeiros autores a representar os Problemas de Escalonamento, utiliza quatro campos: A|B|C|D, em que *A* representa o número de atividades, *B* o número de máquinas, *C* o ambiente de produção e *D* os critérios de otimização.

Graham et al. [24] propõe uma representação mais simples, sendo neste caso α|β|γ, onde α refere-se ao ambiente de produção, β às restrições de processamento, e γ aos critérios de otimização. Esta representação é das mais usadas e diversos autores, como Leung [9], Pinedo [13] e Blazewicz et al. [20], utilizam variações da mesma.

Neste documento será abordada e adotada a nomenclatura apresentada por Pinedo, uma vez que é capaz de representar os diversos detalhes característicos dos diferentes problemas. Como referido, esta resulta de variações na representação de Graham, nomeadamente no que diz respeito à quantidade de restrições do processo de fabrico.

2.3.1. Ambiente de Produção

O campo α indica a quantidade de recursos e a forma como estão dispostos, e contém apenas uma entrada. Pinedo [13] destaca cinco tipos de ambientes que podem ser encontrados:

→ **Máquina Única (1) -** A produção em máquina única é o caso mais simples. Caracteriza-se como um Problema de Sequenciamento, no qual um conjunto de atividades devem ser executadas utilizando um único recurso. Este tipo de problemas é usualmente utilizado na resolução de subproblemas de implementações mais complexas;

→ **Máquinas Paralelas (P_m, Q_m, R_m) -** A produção em máquinas paralelas envolve a existência de várias atividades que podem ser processadas num conjunto de máquinas. Dependendo das características de cada máquina, este tipo de ambiente pode ser dividido em três classes:

- **Máquinas Paralelas Idênticas (P_m)** - As máquinas, dispostas em paralelo, têm velocidades idênticas. Qualquer uma de *m* máquinas é capaz de executar qualquer atividade;
- **Máquinas Paralelas Uniformes (Q_m)** – As máquinas, dispostas em paralelo, têm velocidades diferentes. O tempo de processamento varia de acordo com o desempenho de cada máquina;
- **Máquinas Paralelas não Relacionadas (R_m)** - Representa a existência de máquinas com características distintas, dispostas em paralelo. A velocidade de processamento é dependente da atividade a ser executada.

→ **Linhas de Fabrico ou *Flow Shop* (F_m) -** A produção em *Flow Shop* verifica-se quando existem *m* máquinas, dispostas em série, capazes de executar *n* atividades sequencialmente. As atividades devem possuir a mesma sequência de operações, e, cada uma, após ser processada na máquina *m*, segue para a máquina *m+1,* podendo ou não entrar numa fila de espera antes de ser processada.

- **Linhas de Fabrico Flexíveis ou *Flexible Flow Shop* (FF_c) -** As Linhas de Fabrico Flexíveis são uma generalização dos ambientes *Flow Shop* e Máquinas Paralelas, na qual existem *c* estágios, cada um constituído por uma determinada quantidade de máquinas idênticas em paralelo. Em cada estágio,

uma atividade requer apenas o processamento em uma das máquinas e qualquer uma o pode fazer.

→ **Oficinas de Fabrico ou *Job Shop* (J_m) -** É um tipo de implantação constituído por *m* máquinas, onde cada atividade tem uma sequência específica de processamento e apenas percorre as máquinas que acrescentam valor ao produto final. É um problema completo, o que significa que inclui a afetação dos recursos às atividades e o seu respetivo sequenciamento. Podem existir cenários na qual uma operação seja processada mais do que uma vez num determinado recurso.

  - **Oficinas de Fabrico Flexíveis ou *Flexible Job Shop* (FJ_c) –** Esta implantação é uma generalização dos ambientes *Job Shop* e Máquinas Paralelas, na qual, em vez de *m* máquinas em série, existem *c* centros de trabalho, cada um constituído por uma determinada quantidade de máquinas idênticas em paralelo. Cada trabalho possui a sua própria sequência.

→ **Oficinas Abertas ou *Open Shop* (O_m) –** Representa um cenário com *m* máquinas, no qual as atividades não estão vinculadas a rota específica no sistema, isto é, não existem restrições de ordens obrigatórias para as processar.

2.3.2. Restrições de Processamento

As restrições de processamento, β, especificam as limitações no sistema de produção. É um campo que pode incluir múltiplas entradas simultaneamente, e, de acordo com Pinedo [13], podem ser destacadas as seguintes restrições:

→ **Data de Lançamento (r_j) -** Significa que as atividades apenas podem ser processadas depois da sua data de lançamento, representada por r_j;

→ **Interrupções ($prmp$) -** Representa a possibilidade de interromper o processamento de uma atividade antes da sua conclusão, sem perder o trabalho executado. A atividade pode ser retomada na mesma máquina, ou, no caso de máquinas paralelas, noutra máquina, começando a partir do ponto onde foi interrompida;

→ **Relações de Precedência ($prec$) -** Significa que existe uma ordem de execução entre as atividades, ou seja, uma atividade só pode ser processada após estarem concluídas as atividades que a precedem;

→ **Tempos de *Setup* dependentes da Sequência (s_{jk}) -** Significa que os tempos de *setup* variam de acordo com a ordem em que as atividades são processadas. Se o tempo de preparação entre as atividades *j* e *k* dependerem também da máquina, adiciona-se o índice *i*, ou seja, s_{ijk};

→ **Família de Atividades ($fmls$) -** Representa a existência de famílias de atividades, onde, em cada uma, as atividades podem ter tempos de processamento diferentes, mas serem processadas sem a necessidade de tempo de preparação entre si;

→ **Produção por Lote ($batch(b)$) -** Representa a possibilidade de processar *b* atividades simultaneamente numa máquina. As atividades não necessitam de ter tempos de processamento iguais, e o tempo de conclusão do lote é determinado pelo tempo de processamento da atividade que possui maior duração;

→ **Indisponibilidade das Máquinas ($brkdwn$) –** As avarias são eventos que podem ocorrer e ter como consequência a indisponibilidade temporária das máquinas. Normalmente, utilizam-se distribuições estatísticas para representar a probabilidade de ocorrência de avarias;

→ **Restrições de Elegibilidade das Máquinas (M_j) –** Num ambiente com *m* máquinas em paralelo, M_j representa o conjunto de máquinas que não são capazes de processar a atividade específica *j*;

→ **Permutações ($prmu$) -** Retrata um problema onde não é possível reorganizar a ordem das atividades após serem processadas na primeira máquina, isto é, não é possível fazer permutações na sequência de atividades. É um cenário característico de ambientes *Flow Shop*;

→ **Bloqueio ($block$) -** É um cenário característico de ambientes *Flow Shop*. Uma atividade que acabou de ser processada numa máquina não pode ser libertada se a atividade anterior não tiver concluído o seu processamento na máquina seguinte. Neste caso, a atividade bloqueada também impede ou bloqueia que a próxima inicie o seu processamento na máquina especificada.

→ **Sem Espera (nwt) -** Representa a impossibilidade de uma atividade aguardar entre duas máquinas consecutivas. A atividade só pode entrar em circulação e ser processada quando estiverem presentes todas as condições necessárias para garantir que ela não precisa de esperar por nenhuma das máquinas;

→ **Recirculação ($rcrc$) -** Uma atividade necessita de ser processada numa ou mais máquinas, mais do que uma vez.

2.3.3. Critérios de Otimização

O campo γ das representações de Graham e Pinedo descreve os critérios de otimização, isto é, o objetivo do problema. Neste aspeto, o que cada organização introduz é o reflexo das suas prioridades e do que traduz os objetivos que pretendem alcançar.

Em situações reais de Problemas de Escalonamento, onde várias medidas de desempenho são importantes e precisam de ser otimizadas, visa-se arranjar uma solução de compromisso entre elas para que cada organização consiga garantir a eficiência e eficácia dos seus processos [12].

Entre os diversos critérios mencionados por Pinedo [13] que podem ser incorporados neste campo, alguns dos mais comuns incluem:

→ ***Makespan* (C_{max}) -** Refere-se ao tempo de conclusão de todas as atividades que constituem uma ordem de fabrico. Neste tipo de problemas, o objetivo é minimizar o *makespan*, isto é, reduzir o tempo de conclusão da respetiva ordem de fabrico;

→ ***Maximum Lateness*** **(L_{max}) -** Significa o atraso máximo que uma sequência de atividades apresenta;

→ ***Total Weighted Completion Time*** **($\sum w_j C_j$) -** Representa o tempo total necessário para concluir um conjunto de atividades, em que cada uma possui um peso associado que reflete a sua importância ou prioridade. É comum ser utilizado como um indicador que pode refletir o nível de stock existente, e o objetivo é estabelecer um cronograma que minimize o tempo total de conclusão ponderado;

→ ***Discounted Total Weighted Completion Time*** **($\sum w_j(1 - e^{-rC_j})$) -** O tempo de conclusão ponderado total descontado é uma variação do critério anterior, na qual se acrescentam fatores de desconto que representam a forma como a importância varia ao longo do tempo. O valor de *r* pode variar entre 0 e 1;

→ ***Total Weighted Tardiness*** **($\sum w_j T_j$) –** O objetivo é minimizar os atrasos (ou antecipações) ponderados. É uma medida que quantifica a penalização ou custo proveniente do facto de uma atividade ser concluída após a sua data de conclusão;

→ ***Weighted Number of Tardy Jobs*** **($\sum w_j U_j$) -** Refere-se aos problemas de minimização do número de atividades em atraso, isto é, é um indicador que avalia o número de atividades que não cumpriram a data de entrega**.**

É importante salientar o facto de que várias medidas de desempenho são habitualmente incluídas neste campo por parte de outros autores, o que é um reflexo da complexidade e diversidade inerentes a área de Escalonamento da Produção.

Para uma melhor compreensão, pode ser observado, de seguida, dois exemplos de representações de um Problema de Escalonamento através da nomenclatura de Pinedo.

→ $\mathbf{1 \mid r_j \mid C_{max}}$: Representa um problema em máquina única na qual devem ser processadas n atividades. Cada atividade j possui determinados tempos de chegada, isto é, momentos de lançamento (r_j), e deve ser processada sem interrupção. O objetivo é definir o programa ótimo que minimize o valor do *makespan*;

→ $\boldsymbol{J_2 \mid rcrc, p_{ij} = 1 \mid C_{max}}$: Retrata um ambiente de J*ob Shop* onde cada uma das n atividades tem de ser processada um certo número de vezes em cada uma das duas máquinas (recirculação). Cada atividade tem um tempo de processamento igual a 1, e o objetivo é definir a ordem pela qual as atividades passam pelo sistema, de modo que o *makespan* seja minimizado.

A existência de várias nomenclaturas não deve assim ser vista como uma redundância, mas como uma solução capaz de reunir os diferentes tipos de implantações industriais, as várias restrições e os objetivos que cada empresa tem. Com toda esta variedade de representações à disposição, quem pode beneficiar são as empresas.

2.4. Métodos de Otimização

Numerosos problemas abraçam o âmbito do Escalonamento, e a busca pela melhor solução para alocar recursos e sequenciar atividades nem sempre é fácil e acessível. A complexidade inerente a problemas desta área advém, em parte, da falta de conhecimento ou indisponibilidade de técnicas não enumerativas capazes de proporcionar soluções ótimas, o que confere uma camada suplementar de complexidade ao processo.

Devido à complexidade computacional que requerem em ambientes industriais, é comum classificar os Problemas de Escalonamento como Problemas de Otimização Combinatória. Podem-se distinguir duas classes de problemas, P e NP, referentes a problemas "Fáceis" e problemas "Difíceis", respetivamente [12].

Num problema de classe P, o tempo de execução está limitado por um polinómio e o problema pode ser resolvido por uma DTM (*Deterministic Turing Machine*), enquanto um problema NP (*Non-Deterministic Polynomial Time*) pode ser resolvido em tempo polinomial por uma NDTM (*Non-Deterministic Turing Machine*), ou seja, é inviável testar todas as soluções em tempo computacional útil [12, 20, 25].

Com o intuito de solucionar este tipo de problema, têm emergido algoritmos que demonstram a capacidade de fornecer soluções satisfatórias em períodos de processamento computacional relativamente reduzidos. A Figura 3 apresenta os principais métodos de otimização para Problemas de Otimização Combinatória mencionados em [26], que divide em "Métodos Exatos" e "Métodos Aproximados".

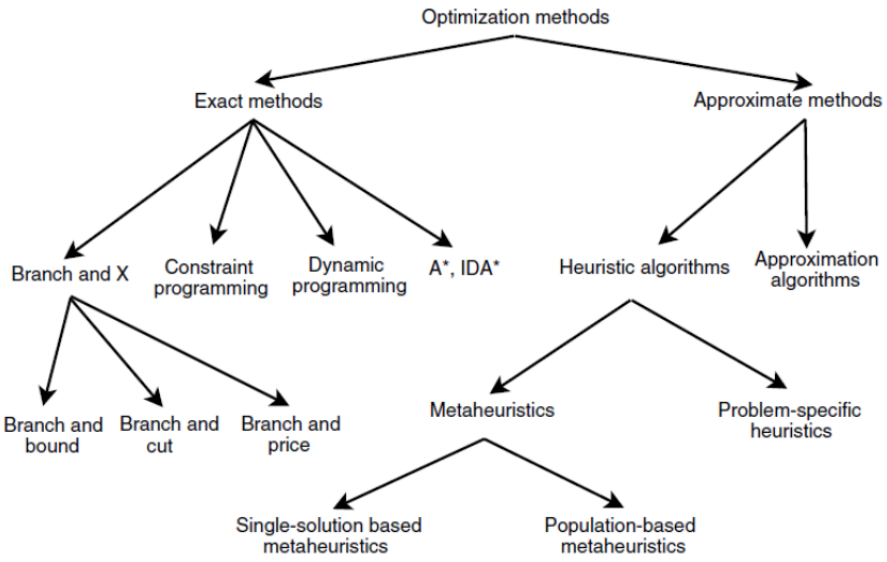


Figura 3- Métodos de otimização [26]

2.4.1. Métodos Exatos

Os métodos exatos caracterizam-se por procederem à exploração completa e exaustiva do espaço de soluções para encontram a solução ótima para um problema, com base no critério de otimização definido. O problema resolve-se subdividindo-se em partes mais simples [22].

Segundo Pinedo [13] e Branke et al. [16], estes métodos apresentam um bom desempenho em problemas de pequena dimensão. Quando o grau de complexidade é elevado, a complexidade cresce exponencialmente com a dimensão do problema, o que dificulta a capacidade de encontrar uma solução ótima.

De acordo a Figura 3, entre as técnicas de otimização exatas é importante destacar o algoritmo *Branch and Bound*, a Programação Dinâmica, a Programação por Restrições, o A^+ e o IDA^+.

Branch and Bound

É o método exato mais conhecido de uma família de técnicas composta pelos métodos *Branch and Bound, Branch and Cut e Branch and Price*. Caracteriza-se como sendo um algoritmo que decompõe repetidamente um problema inicial através da criação e exploração de uma árvore de pesquisa dinamicamente. O nó raiz representa o problema inicial, os nós internos são subproblemas do problema inicial, e os nós folha são soluções possíveis [27]. Cada nó correspondente a uma solução parcial tem associado a si um valor intrincado de limite inferior que determina se o nó pode ser "podado", reduzindo assim a região de pesquisa na árvore e encurtando a duração do processo computacional [28].

Os métodos utilizados para dividir o problema em subproblemas representam as etapas de "*branching",* enquanto a fase de *"bounding"* consiste na eliminação dos subproblemas [28].

A pesquisa e aplicabilidade deste método de otimização têm crescido nas últimas décadas, o que lhe confere um papel relevante na resolução de uma ampla gama de desafios e problemas relacionados com o Escalonamento. Ahn & Kim [29] recorreram ao algoritmo *Branch and Bound*, apresentado na Figura 4, com o objetivo de obter uma programação ótima de Sistemas de Fabrico Flexíveis, e os resultados experimentais revelaram que este método superou, de modo notável, não somente a formulação matemática, mas também outros algoritmos preexistentes, em várias instâncias do problema. Além disso, demonstrou a sua eficácia na capacidade de lidar com vários layouts de sistemas flexíveis e na redução significativa do tempo computacional e dos espaços de pesquisa.

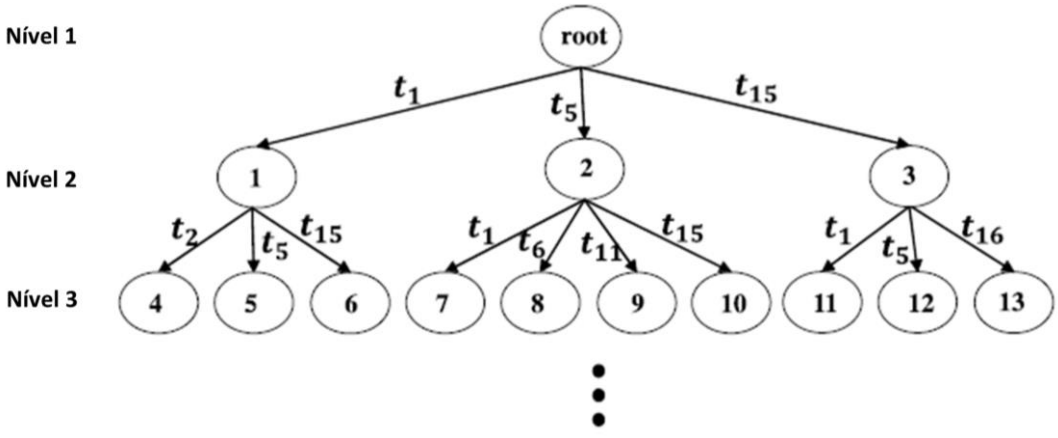


Figura 4- Árvore de pesquisa desenvolvida na aplicação do método *Branch and Bound* [29]

Programação Dinâmica

Este método consiste na subdivisão do problema original em subproblemas de menor dimensão. A cada passo, é feita a seleção de um conjunto de decisões que viabilizam a obtenção do melhor resultado em relação à função objetivo.

De acordo com Pinedo [13], a Programação Dinâmica é uma abordagem de otimização enumerativa que, em cada iteração, emprega a informação adquirida nas soluções dos problemas anteriores para determinar a solução ótima de cada subproblema, que é mais complexo que os que foram previamente resolvidos. Baseia-se no Princípio de Otimalidade de Bellman [30], que refere que "Uma política ótima tem a propriedade de, independentemente do estado e das decisões iniciais, as decisões restantes devem constituir uma política ótima em relação ao estado resultante das decisões anteriores" [31].

Apesar da Programação Dinâmica ser um método que, ao longo do tempo, tem sido reinventado pelos investigadores na área do Escalonamento, a sua utilização prática não é das mais comuns, quando comparada com a de outros métodos exatos, porque a maioria dos estudos se concentram em abordagens e soluções teóricas, negligenciando, em algumas circunstâncias, a análise prática dos modelos associados a este tipo de algoritmo. Para além disso, muitos Problemas de Escalonamento são "NP-Hard", o que faz com que problemas de natureza recursiva, como aqueles abordados pela Programação Dinâmica, sejam mais difíceis de formular devido às complexas estruturas das suas formulações, que necessitam de ser matemática e computacionalmente viáveis [32].

Programação por Restrições

A Programação por Restrições é um método modelado em torno de conceitos de árvore de pesquisa e de um conjunto de variáveis limitadas por restrições [33]. As variáveis podem assumir os valores num domínio finito, enquanto as restrições podem apresentar formas matemáticas ou simbólicas [26, 34].

O principal objetivo da abordagem é encontrar uma solução que satisfaça todas as restrições, isto é, uma solução viável. A conexão entre restrições e variáveis é realizada através de uma técnica de propagação, também conhecida como "filtragem", onde, em cada restrição, avalia-se os valores das variáveis e remove-se, dos seus domínios, aqueles que não cumprem os requisitos, ou seja, que não podem conduzir a soluções viáveis [26, 35].

As características deste algoritmo propiciam a sua ampla aplicação num leque diversificado de desafios nos últimos anos, nomeadamente em contextos relacionados com o Escalonamento. Esta particularidade deve-se à capacidade de o algoritmo proporcionar formulações de carácter flexível, conciso e simplificado, de, quando contrastado com outros métodos exatos, ser uma técnica que não precisa de muitos ajustes para ser implementada com outra, e à sua aptidão para ser aplicado em problemas com restrições diferentes, uma vez que cada restrição possui a peculiaridade de poder ter o seu próprio algoritmo de filtragem [35].

Família de Métodos A^*

A^* é uma técnica de enumeração implícita desenvolvida para resolver Problemas de Otimização Combinatória [26]. Como algoritmo de pesquisa, tem o objetivo de encontrar qual é o caminho mais curto entre o estado (ou nó) inicial e o estado final. O processo de decisão sobre qual é o próximo nó a seguir, n, engloba determinar a distância (ou custo) entre o nó inicial e o nó n, e estimar, através de uma Heurística, a menor distância entre o nó n e o nó final, este último representativo do objetivo do problema [36].

Por sua vez, IDA^* *(Iterative-Deepening A^*)* é uma variação linear do algoritmo A^*, que se apresenta como mais vantajoso quando o problema é limitado pela memória. Enquanto o algoritmo A^* armazena todos os nós gerados, o IDA^* apenas ocupa a memória com os nós que fazem parte do caminho atual [37]. Um exemplo da aplicação do método IDA^* num problema de realocação de contentores, com a descrição pormenorizada do processo de pesquisa, pode ser analisado em [38].

Em suma, os métodos de otimização exatos assumem-se como algoritmos importantes, no entanto, devido às suas limitações, a sua aplicação é reduzida quando existem técnicas mais eficientes, o que faz com que sejam adequados para problemas com instâncias pequenas.

2.4.2. Métodos Aproximados

Na resolução de problemas complexos, nos quais se incluem os Problemas de Escalonamento, enumerar todas as soluções e explorar todo o espaço de soluções possíveis em tempo útil torna-se impossível, e nem sempre é fácil encontrar uma solução ótima. Para solucionar este problema, utilizam-se métodos aproximados que são capazes de encontrar soluções bastantes satisfatórias em intervalos de tempo aceitáveis, mesmo quando aplicados a instâncias de problemas de grande dimensão [16, 26].

Fuchigami & Rangel [39], num estudo sobre as aplicações do Escalonamento em vários segmentos da indústria, verificaram que 71,19% dos artigos analisados aplicaram Heurísticas e Meta-heurísticas. O facto de a utilização de métodos aproximados ser superior à das técnicas exatas evidencia assim que, em algumas situações, o custo necessário para "obter a solução ótima" não se justifica, o que faz com que sejam aplicados métodos aproximados.

De acordo com o representado na Figura 3, nos métodos aproximados podem-se incluir duas classes de algoritmos: as Heurísticas e as Técnicas de Aproximação. As Heurísticas podem-se dividir em Heurísticas Específicas, que são concebidas para lidar com as características únicas de um determinado problema, e Meta-Heurísticas, que tentam melhorar uma programação previamente construída através da aplicação de algum tipo de conhecimento. Entre todas as técnicas mencionadas, serão abordadas de seguida as Técnicas de Aproximação, as Heurísticas Construtivas, a Pesquisa Local e as Meta-Heurísticas.

Técnicas de Aproximação

As Técnicas de Aproximação são métodos aproximados para os quais existe um limite máximo para a diferença entre a solução calculada e o ótimo, o que significa que, à partida, fornecem

garantias sobre a aproximação. Sendo s a solução ótima e ϵ o fator de distância máxima, a solução obtida com a técnica, α, não deve ser inferior ao produto $s \times \epsilon$, $se\ \epsilon > 1$, ou superior $se\ \epsilon < 1$ [26].

Para resolver problemas como *Bin Packing* e *Traveling Salesman Problem* (TSP), existem as Técnicas de Aproximação, mas também Heurísticas e Meta-Heurísticas. Relativamente às Técnicas de Aproximação, o estudo para resolver o TSP, considerado NP-Hard, sempre foi muito intensivo, o que permitiu a existência de diversas soluções. Uma das mais conhecidas, proposta por Christofides, atribui o fator de aproximação de $\frac{3}{2}$. Outros desenvolvimentos e tentativas de melhoria deste problema podem ser consultadas de forma mais detalhada em [40].

Heurísticas Construtivas

As Heurísticas Construtivas são métodos que criam soluções viáveis, partindo de uma solução vazia e progredindo iterativamente até construir uma solução completa. Cada solução é formada através da adição de elementos, isto é, atribuindo valores às variáveis de decisão [41].

As Heurísticas Construtivas também são chamadas de "Heurísticas Gulosas", ou Heurísticas Míopes, porque, em cada ponto da construção da solução, elegem a alternativa mais vantajosa naquele momento, sem considerar previsões sobre eventuais implicações futuras nem revisões de decisões anteriormente tomadas [26].

Normalmente, estes métodos conseguem obter resultados satisfatórios em alguns minutos. Esta rapidez é crucial para satisfazer requisitos em tempo real e lidar com problemas de grande escala. Adicionalmente, são Heurísticas que, em diversos casos, conseguem oferecer soluções iniciais eficazes para outros métodos aproximados, como as Meta-Heurísticas [42]. No entanto, possuem limitações em relação à natureza específica da sua aplicação. Branke et al. [16] mencionam que são métodos de solução específicos do problema, ou seja, têm de ser concebidos para o problema em causa. Quando a situação exige o desenvolvimento de Heurísticas sofisticadas, o processo torna-se muito extenso, requerendo uma quantidade significativa de conhecimento especializado e um incremento proporcional de esforço porque é necessário submeter as Heurísticas a sucessivas iterações de testes, proceder a modificações e, subsequentemente, testá-las novamente até que alcancem resultados consonantes com os requisitos inerentes ao problema em foco.

Um caso particular das Heurísticas Construtivas são as Regras de Despacho ou Prioridade, que são utilizadas para selecionar a atividade seguinte a ser processada num conjunto de atividades que aguardam num recurso para serem executadas [43]. A determinação do índice de prioridade depende das características da atividade, como o tempo de processamento, e do recurso, como o tempo médio de processamento na fila de espera do recurso [44, 45].

As Regras de Prioridade podem ser classificadas como estáticas ou dinâmicas. Nas regras estáticas, como *Shortest Processing Time*, as prioridades das atividades permanecem inalteradas durante a sua execução, ou seja, não dependem do tempo, mas apenas das características das máquinas ou das atividades. Nas regras dinâmicas, as prioridades das atividades dependem do tempo, o que significa que podem ser ajustadas com base em condições dinâmicas, como acontece com a regra da folga mínima*, Minimum Slack* [13].

No que diz respeito à informação em que cada as regras se baseiam, estas podem ser categorizadas como locais ou globais. Numa regra local, a prioridade de uma atividade é determinada pelas informações da fila de espera da máquina onde aguarda para ser executada. Numa regra global, como a *Longest Alternate Processing Time* para *Open Shop* com duas máquinas, podem ser utilizadas informações relativas a outras máquinas para obter o índice de prioridade de uma atividade. Caso uma regra resulte da combinação de duas Regras de Prioridade, podem-se designar como regras compostas [13].

A literatura relativa à área do Escalonamento apresenta uma extensa variedade de Regras de Prioridade. Na Tabela 1 são apresentadas algumas das regras mais populares, neste caso mencionadas em [13, 18].

Tabela 1- Regras de Prioridade (Adaptado de [13, 18])

CP	*Critical Path*
EDD	*Earliest Due Data*
ERD	*Earliest Release Date*
FCFS	*First Come First Served*
LAPT	*Longest Alternate Processing Time*
LFJ	*Least Flexible Job*
LNS	*Longest Number of Successors*
LPT	*Longest Processing Time*
LWKR	*Least Work Remaining*
MST	*Minimum Slack Time*
MWKR	*Most Work Remaining*
SIRO	*Service in Random Order*
SPT	*Short Processing Time*
SQ	*Shortest Queue*
SQNO	*Shortest Queue at the Next Operation*
SST	*Shortest Setup Time*
WSPT	*Weighted Shortest Processing Time*

2.4.3. Conceitos Básicos

Para perceber a base do funcionamento de técnicas que necessitam de representar (ou codificar), avaliar e manipular soluções, como a Pesquisa Local e as Meta-Heurísticas, é fundamental abordar os conceitos de Codificação e Estrutura de Vizinhança (EV).

Codificação

A codificação desempenha um papel crucial na resolução dos Problemas de Otimização. A essência do processo de codificação reside na representação de soluções para um determinado problema numa forma que seja compreensível e manipulável por algoritmos, como Meta-Heurísticas, que, no seu funcionamento, necessitam de gerar, avaliar e manipular soluções candidatas com base em critérios específicos do problema.

A codificação influencia a eficiência e a eficácia de qualquer algoritmo e deve ser, por isso, adequada e relevante para o Problema de Otimização. Neste aspeto, é importante considerar os aspetos relacionados com o algoritmo que vai resolver o problema, como o modo como a solução será avaliada e como será gerada a vizinhança da mesma, ou seja, que operadores serão aplicados [46]. É assim importante representar uma solução de uma maneira que permita pesquisar o espaço de soluções de forma adequada, sem adicionar constrangimentos que aumentem a sua dimensão, dificultem a descoberta de soluções de qualidade e/ou aumentem o tempo necessário para resolver o problema [26, 47].

Para resolver uma ampla variedade de Problemas de Otimização, Talbi [26] menciona duas classes de codificação: Codificação Linear e Codificação Não Linear. A Codificação Linear possui uma ampla aplicação e inclui, na sua constituição, a Codificação Binária, a Codificação Discreta, a Codificação de Permutação e a Codificação baseada num Vetor de Valores Reais [26, 47].

Na Codificação Binária é associado um valor binário - $\{0, 1\}$, a cada variável de decisão. Uma solução é codificada como um vetor de bits, ou seja, por um vetor de variáveis binárias, o que é comum em muitos Problemas de Otimização, como o popular *Knapsack Problem*, onde a Codificação Binária pode indicar se o objeto se encontra na mochila, ou não [47, 48]. Quando não se usam apenas dois símbolos na codificação e esta é generalizada em valores discretos, é considerada uma Codificação Discreta (ou Vetor de Valores Discretos). Este tipo de codificação representa as variáveis de decisão por números inteiros, o que é característico de Problemas de Otimização Combinatória nos quais as variáveis podem assumir um número finito de valores [26, 47].

A Codificação de Permutação caracteriza-se por representar uma ordem/sequência das variáveis de decisão, $\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$. É comum ser utilizada para representar as soluções em Problemas de Sequenciamento, onde uma codificação pode representar a sequência de processamento de operações, e em Problemas de Roteamento, como o TSP, na qual a codificação representa a sequência de cidades a visitar numa rota [47, 48]. Por sua vez, a última classe de codificação é representada através de um Vetor de Valores Reais, no qual cada elemento é representado por um número real. Em diversos problemas, este é um tipo de codificação que é capaz de ultrapassar algumas das limitações apresentadas pelas outras classes de codificação, e, no caso específico dos Problemas de Otimização não Lineares, é considerada uma das mais utilizadas para representar as soluções [26, 47].

A Figura 5 apresenta um exemplo de cada um dos quatro tipos de Codificação mencionados.

Figura 5- Tipos de Codificação para Problemas de Otimização

Naturalmente, em função das características e necessidades do problema, existem outras classes de codificação, como as apresentadas em [26] e comuns dentro da comunidade da computação evolutiva.

Relativamente às Codificações não Lineares, estas são geralmente estruturas mais complexas, baseadas principalmente em estruturas de grafos. Na Codificação em Árvore, que é a que apresenta maior aplicação, uma solução é representada por uma árvore de alguns objetos. Quando aplicada, por exemplos, na Programação Genética, a árvore pode codificar uma expressão aritmética [26].

Estruturas de Vizinhança

No contexto da otimização, a noção de vizinhança desempenha um papel fundamental e central em algoritmos como as Meta-Heurísticas porque permite que explorem o espaço de solução na procura por soluções de melhor qualidade [49]. As Estruturas de Vizinhança permitem definir e explorar a proximidade de uma solução, e determinam o modo como uma Meta-Heurística se desloca no espaço de soluções. Isto engloba determinar como é que a solução atual pode ser modificada para gerar outras soluções dentro dessa vizinhança, o que corresponde, por exemplo, à definição de um conjunto específico de operações ou movimentos que podem ser aplicados a essa mesma solução [50, 51].

As Estruturas de Vizinhança podem variar em função do problema a resolver. Quando são desenvolvidas para resolver um problema em específico, a complexidade da Estrutura de Vizinhança pode ser elevada, no entanto, também existem estruturas que se destacam por apresentarem mecanismos mais simples, o que permite as suas aplicações a diversos problemas [50, 52]. A construção de uma solução vizinha através de operações de trocas de elementos é uma forma típica para os Problemas de Permutação, na qual se incluem os Problemas de Escalonamento [47]. Na Figura 6 são abordadas quatro Estruturas de Vizinhança comuns de Permutação – *Transpose, Swap, Insert, Reverse*. No primeiro caso, *Transpose*, as soluções são obtidas através da permutação da posição de dois elementos adjacentes da solução [52, 53]. Por exemplo, a solução $\{1,2,3,4,5,6,7,8\}$ transforma-se na solução $\{1,2,3,4,5,7,6,8\}$ devido à troca dos elementos $\{6,7\}$. A segunda situação, *Swap*, envolve a troca de posições entre elementos na solução [54, 55]. Por exemplo, a solução $\{1,2,3,4,5,6,7,8\}$ dá origem à solução $\{1,6,3,4,5,2,7,8\}$ pela troca entre os elementos 2 e 6. O terceiro caso, *Insert*, representa uma situação onde é selecionado um elemento aleatoriamente para ser removido da sua posição atual e inserido numa posição diferente da sequência, como acontece com a transformação da solução $\{1,2,3,4,5,6,7,8\}$ na solução $\{1,2,4,5,6,3,7,8\}$ [54]. Na quarta Estrutura, *Reverse*, seleciona-se dois elementos diferentes e, para além de estes serem trocados, também se inverte a sequência dos números entre eles [56, 57]. Por exemplo, na solução $\{1,2,3,4,5,6,7,8\}$, se os elementos selecionados foram o 2 e o 5, a solução vizinha é $\{1,5,4,3,2,6,7,8\}$.

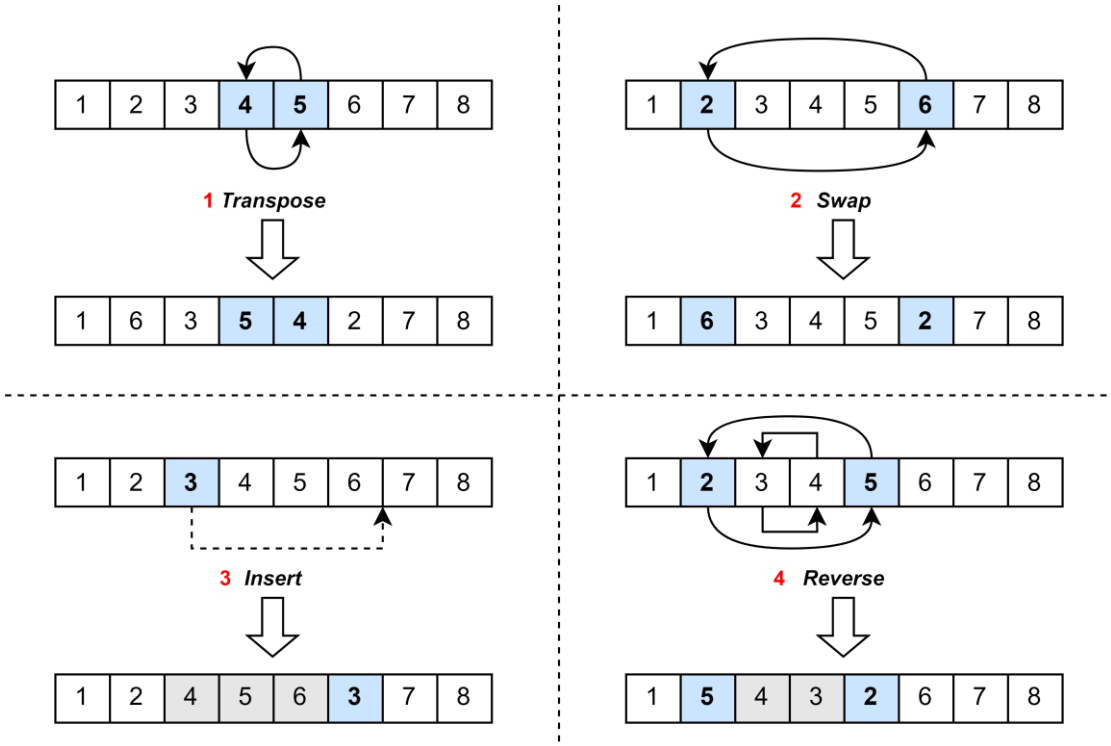


Figura 6- Estruturas de Vizinhança comuns em vários tipos de problemas

A escolha da Estrutura de Vizinhança apropriada é crucial para o desempenho e eficácia de um algoritmo de otimização. Estruturas bem projetadas podem permitir uma exploração do espaço de soluções de forma eficiente. Para além disso, a combinação de diferentes estruturas ou as suas adaptações para problemas em específico podem melhorar ainda mais a capacidade de o algoritmo encontrar soluções de alta qualidade, neste caso ótimas ou mesmo próximas da ótima em problemas com complexidade acrescida.

2.4.4. Pesquisa Local

A Pesquisa Local (LS, de *Local Search*) é um algoritmo que pesquisa soluções no espaço de soluções candidatas. A partir de uma solução inicial, que pode ser gerada aleatoriamente ou através de outros métodos como regras de prioridade, pesquisa o espaço de vizinhança com o objetivo de procurar soluções com melhor desempenho. Caso esta seja encontrada, substitui a atual e todo o processo repete-se novamente. Se todos os vizinhos forem analisados e apresentarem um desempenho inferior ao da solução atual, o processo termina [26].

A Tabela 2 apresenta o algoritmo geral de LS. Neste caso s representa a solução atual, s_0 a solução inicial, $N(s)$ é o conjunto de soluções vizinhas de s, s_i é cada solução de $N(s)$ analisada em cada iteração, e s' é a melhor solução vizinha de s.

Tabela 2- Algoritmo genérico representativo da Pesquisa Local (adaptado de [26, 58])

Gerar a solução inicial $s = s_0$
While Critério de Interrupção não Ocorrer **Do**
Gerar $N(s)$
If $s_i \in N(s)$ não for melhor que s **Then**
Stop
Selecionar a melhor solução, $s' \in N(s) \Rightarrow s = s'$
EndWhile
Output: s

Apesar de ser um método fácil de desenvolver e implementar, e de permitir a exploração da vizinhança da solução inicial, a Pesquisa Local também possui desvantagens associadas. Talbi [26] refere que é um algoritmo muito sensível à solução inicial, o que pode ter como consequência uma grande variabilidade da qualidade das soluções para alguns problemas. Xhafa et al. [59] acrescentam o facto de as soluções convergirem para um ótimo local, isto é, apenas para a melhor solução de uma vizinhança, e não para a solução com melhor desempenho, conhecida como ótimo global. Para ultrapassar esta situação e conduzir a pesquisa para outras vizinhanças onde possa existir o ótimo global, é comum partir de soluções iniciais diferentes, alterar o método estabelecido para manipular as soluções (estrutura de vizinhança), ou utilizar uma Meta-Heurística, que, no seu desenvolvimento, possui estruturas e mecanismos capazes de ultrapassar ótimos locais.

Ao longo do tempo, a aplicabilidade deste método sempre foi ampla. Em [60] foi proposto um algoritmo de pesquisa harmónica discreta de agrupamento baseado em Pareto para o Escalonamento Flexível e Multiobjectivo em ambiente *Job Shop*, na qual utilizaram, numa fase intermediária, dois métodos de Pesquisa Local para aumentar a capacidade de exploração. Em ambos os critérios, o *Makespan* e a data de entrega (relação *Earliness-Tardiness*), o procedimento desenvolvido permitiu definir a sequência de operações ótima em cada máquina.

Relativamente às Meta-Heurísticas, considerando que são um dos focos desta dissertação devido à sua relação com o ML, estas serão analisadas com maior detalhe de seguida.

2.5. Meta-Heurísticas

A palavra Meta-Heurística combina o prefixo grego “Meta”, que representa “para além” ou “num nível superior”, com o termo “Heurística”, derivado do verbo grego *heuriskein*, que significa “encontrar”. Desta forma, as Meta-Heurísticas podem ser definidas como algoritmos que combinam Heurísticas num quadro mais amplo e abrangente [61, 62].

Não obstante a sua natureza ubíqua, uma Meta-Heurística, como método científico para a resolução de problemas, é um fenómeno moderno [63]. A sua definição formal fundamenta-se numa variedade de definições de diferentes autores com base na contribuição de Glover [64], que foi o primeiro autor a utilizar a expressão, em 1996 [2]. Osman e Laporte [65] referem que uma Meta-Heurística é um método que orienta uma Heurística subordinada no sentido de

explorar o espaço de pesquisa. Para produzir soluções de alta qualidade, pode manipular, em cada iteração, uma solução completa (ou incompleta) ou um conjunto de soluções [66].

Para a comunidade científica, as Meta-Heurísticas são uma alternativa viável, e muitas vezes superior, aos métodos exatos porque, apesar de não garantirem soluções ótimas, são capazes de oferecer um melhor compromisso entre a qualidade da solução e o tempo de computação. Em [67], Blum e Roli enunciam as características que podem ser associadas às Meta-Heurísticas:

- → Guiam o processo de pesquisa visando explorar eficientemente o espaço de pesquisa. As Meta-Heurísticas mais avançadas podem fazê-lo usando a memória e a experiência;
- → Durante o processo de pesquisa, são capazes de incorporar mecanismos para evitar ficar presas em ótimos locais (melhores soluções numa vizinhança). O objetivo é encontrar o ótimo global, ou seja, a solução com melhor desempenho do problema;
- → Podem ser aplicadas a qualquer tipo de problema sem necessitarem de efetuar quaisquer alterações especiais na estrutura do algoritmo;

Neste domínio, Mirjalili et al. [68] acrescentam dois aspetos que explicam os motivos pelos quais estes métodos se tornaram tão prevalentes nas últimas duas décadas. Em primeiro lugar, destaca-se a simplicidade intrínseca das suas conceções, uma vez que são inspiradas em conceitos simples, como fenómenos físicos ou comportamentos de animais. Essa simplicidade não apenas agiliza os processos de aprendizagem e aplicação, como também confere uma base conceitual acessível. Em segundo, enfatiza-se o mecanismo livre de derivação, no qual, a partir de uma solução aleatória, otimizam o problema estocasticamente. Com essa abordagem, prescinde-se do cálculo de derivadas nos espaços de pesquisa para encontrar o ótimo, tornando assim as Heurísticas particularmente adequadas para lidar com problemas reais com informações dispendiosas ou desconhecidas.

O cerne de um algoritmo Meta-Heurístico é formado pelos conceitos de Intensificação e Diversificação, que são responsáveis por orientar a pesquisa no espaço de soluções. A intensificação refere-se à fase de exploração do processo de pesquisa onde o algoritmo reduz o espaço para explorar regiões promissoras, concentrando-se assim em refinar e melhorar a solução atual. A diversificação representa uma exploração da totalidade do espaço de soluções com o objetivo de promover a descoberta de soluções globalmente melhores e evitar que o algoritmo fique preso em ótimos locais [63, 67].

No âmbito das Meta-Heurísticas, é fundamental estabelecer um equilíbrio entre a intensificação e a diversificação. Uma intensidade excessiva pode resultar na retenção do algoritmo em ótimos locais, enquanto uma diversidade excessiva pode dificultar a convergência para soluções de qualidade [69].

A Figura 7 apresenta os conceitos apresentados anteriormente. Na intensificação, a pesquisa intensifica-se na área a tracejado, que corresponde ao espaço de soluções próximo de uma solução promissora. Na diversificação, as diversas setas ao longo de todo o espaço de pesquisa evidenciam a exploração do mesmo. Por sua vez, o ótimo global, a solução com desempenho inferior, é representado a azul, enquanto os ótimos locais a cor-de-laranja.

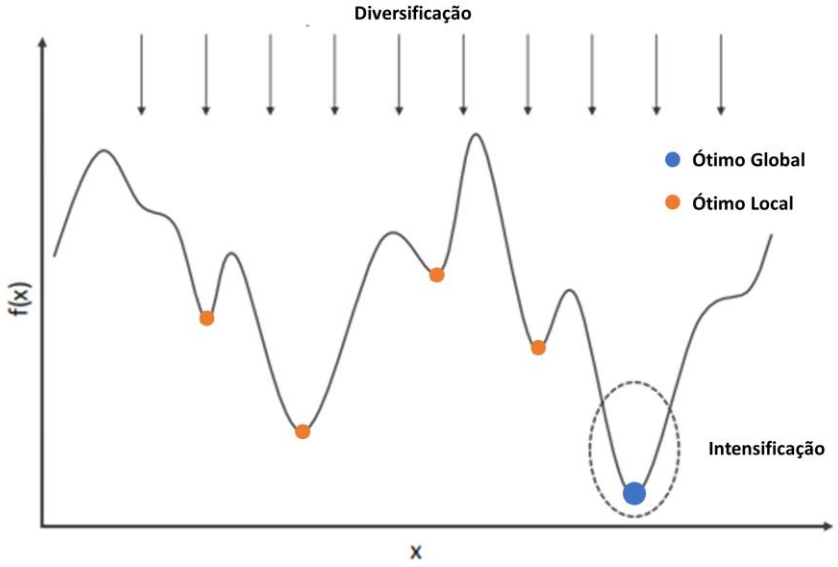


Figura 7- Intensificação, Diversificação e Ótimos Locais e Globais (adaptado de [70])

Dependendo das características selecionadas, existem diversas formas de classificar as Meta-Heurísticas. A Tabela 3 apresenta alguns tipos de classificações utilizados frequentemente.

Tabela 3- Diferentes formas de classificação das Meta-Heurísticas

Fonte de Inspiração/Origem do algoritmo [63, 71]	
Inspirada na Natureza	Não Inspirada na Natureza
Meta-Heurísticas inspiradas em processos naturais. A natureza funciona como fonte de conceitos, mecanismos e princípios para o desenvolvimento dos algoritmos.	Meta-Heurísticas que não são derivadas de processos da natureza. Utilizam técnicas computacionais para explorar o espaço de soluções.
Exemplos: *Simulated Annealing, Particle Swarm Optimization* (PSO)*, Firefly Algorithm, Ant Colony Optimization* (ACO)*, Grey Wolf Optimizer, Whale Optimization Algorithm*	Exemplos: *Tabu Search, Iterated Local Search, Greedy Randomized Adaptive Search Procedure*

Número de soluções utilizadas [72, 73]	
Baseada numa Solução Única	Baseada em População
Partem de uma solução e procuram as melhores soluções na vizinhança. Evitam ficar presas em ótimos locais de forma estocástica, intensificando a pesquisa em regiões locais.	Utilizam as características da população para controlar a pesquisa, manipulando várias soluções em simultâneo. Ao utilizar múltiplas soluções, realizam uma exploração mais ampla do espaço de soluções.
Exemplos: SA, TS, *Iterated Local Search, Greedy Randomized Adaptative Search Procedure, Variable Neighborhood Search*	Exemplos: *Artificial Bee Colony, Genetic Algorithm* (GA), *Evolution Strategies,* ACO*, PSO, Cultural Algorithms*

Modo como é tratada a informação sobre soluções passadas [26, 67]

Com Memória	Sem Memória
Utilizam estruturas de memória para armazenar informações sobre soluções anteriores encontradas, orientando o algoritmo durante o processo de pesquisa.	Em cada iteração, baseiam-se apenas na informação disponível no momento, ou seja, sem considerar o histórico de soluções passadas.
Exemplos: TS, ACO, *Genetic Algorithm with Elitism, Memetic Algorithm*	Exemplos: SA*, Local Search, Greedy Randomized Adaptative Search Procedure, Evolution Strategies*

Ao longo do tempo têm surgido novos estudos, nos quais os investigadores, para além de se esforçarem no melhorar dos métodos existentes e/ou na combinação de diversos algoritmos para otimizar as suas capacidades individuais, também se dedicam ao desenvolvimento de novos algoritmos. Para os enquadrar na literatura atual, as categorias atuais são subdivididas em subcategorias, como acontece, por exemplo, com a inspiração na natureza, que é uma das classificações mais antigas. Rajakumar et al. [74] e Mohamed et al. [75] classificam os algoritmos inspirados na natureza em quatro grupos:

- → Técnicas Evolucionárias, que são inspiradas na biologia. Uma população inicial aleatória evolui ao longo das gerações para produzir novas soluções através do cruzamento e mutação, eliminando as piores soluções;
- → Técnicas de Inteligência de Enxame, baseadas no comportamento de insetos;
- → Técnicas baseadas na Física, inspiradas nas regras que regem um fenómeno natural;
- → Técnicas relacionadas com os Humanos, que se baseiam nas atividades físicas e não físicas que os caracterizam.

Por sua vez, Molina et al. [76] utilizam seis categorias, que são os Algoritmos Evolutivos baseados em Reprodução, baseados em Inteligência de Enxame, baseados em Físico-Química, baseados no Comportamento Social Humano, baseados em Plantas, e o resto é considerado como Diverso. Apesar de não diferir substancialmente da classificação apresentada anteriormente, introduz novos conceitos que assumem uma elevada importância.

Outras formas de classificação de Meta-Heurísticas que complementam a informação disponibilizada na Tabela 3 podem ser observadas nos estudos apresentados por Blum e Roli [67], Ezugwu et al. [71] e Rajwar et al. [73]. Uma revisão mais prática e meticulosa acerca de classificações empregadas ao longo do tempo foi desenvolvida por Stegherr et al. [77].

Na Secção 2.5.1 e na Secção 2.5.2 serão abordadas duas Meta-Heurísticas, *Simulated Annealing* e *Tabu Search*, respetivamente, que se destacam pela sua aplicação recorrente em diversas categorias de problemas, incluindo os Problemas de Escalonamento.

2.5.1. Simulated Annealing

Simulated Annealing é uma Meta-Heurística introduzida em 1983 por Kirkatrick et al. [78], baseada no método de *Metropolis-Hasting*. Foi proposta como uma técnica geral de otimização para vários problemas, onde se incluem os Problemas de Otimização Combinatória.

O algoritmo SA é inspirado no processo de recozimento natural dos sólidos, em que um metal é aquecido a altas temperaturas, e, posteriormente, é arrefecido lentamente até atingir a configuração mais regular da rede cristalina. À medida que a temperatura diminui, os átomos presentes no material fundido procuram estabilizar-se no estado de energia mais baixo, no entanto, nem sempre conseguem alcançar a posição ótima devido à interferência de outros átomos, que podem ocupar posições que impedem esse processo [2]. De maneira análoga, o algoritmo inicia a partir de um estado inicial de alta energia (solução inicial), e, de maneira progressiva, reduz a temperatura, um parâmetro característico, até atingir um estado de energia mínima (a solução ótima). Durante este processo, assim como os átomos se podem mover para posições que diminuem a energia, as alterações às soluções também são aceites, caso as melhorem ou piorem [79, 80].

Trata-se de uma Meta-Heurística que parte de uma solução inicial, normalmente gerada aleatoriamente. Em cada iteração, uma nova solução é gerada através de um mecanismo de vizinhança, e comparada com a solução atual. Caso a nova solução, conhecida como solução candidata, apresente um desempenho superior, esta substitui a solução atual. Se for inferior, substitui a solução atual, mas com uma determinada probabilidade. Aceitar soluções piores com uma determinada probabilidade funciona assim como uma estratégia para evitar ótimos locais e convergir para um ótimo global [72, 81].

A probabilidade de aceitação de uma solução com pior desempenho que a solução atual é calculada através da Equação (4), e depende da Temperatura Atual, T, e do parâmetro $\Delta\mathrm{E}$, que é a diferença entre a função objetivo da solução candidata, $\mathrm{f}(\mathrm{s}')$, e da solução atual, $\mathrm{f}(\mathrm{s})$ [26].

$$P(\Delta E, T) = e^{-\frac{\Delta E}{T}} \tag{4}$$

Para cada temperatura T, o processo de gerar novas soluções e analisá-las é feito durante L vezes, sendo L conhecido como *Epoch Length* (EL). Este parâmetro representa o número de iterações executadas para cada valor de T, ou o número de soluções aceites até T diminuir [82].

A Tabela 4 apresenta o algoritmo genérico do SA. Neste caso s refere-se à solução atual, s_0 à solução inicial, s' é a solução candidata, T a Temperatura atual, T_i a Temperatura inicial.

Tabela 4- Algoritmo representativo do *Simulated Annealing* (adaptado de [26])

Input: Esquema de Arrefecimento
Gerar a solução inicial $\mathrm{s} = \mathrm{s}_0$
Inicializar a temperatura $\mathrm{T} = \mathrm{T}_\mathrm{i}$
While Critério de Interrupção não Ocorrer **Do**
    While L **Do**
        Gerar aleatoriamente s'

$\Delta E = f(s') - f(s)$
If $\Delta E < 0$ **Then**
s=s'
Else
$s = s'$ com probabilidade $P(T, \Delta E) = e^{-\frac{\Delta E}{T}}$
EndWhile
$T = g(T)$
EndWhile
Output: Melhor Solução

O pressuposto subjacente ao SA é que, à medida que o tempo avança e o algoritmo progride, devemos aproximar-nos de um ótimo global e ser menos propensos a aceitar desvios desse ótimo [83]. Por outras palavras, inicialmente, quando as temperaturas são mais elevadas, uma solução com desempenho inferior apresenta maior probabilidade de ser aceite, o que permite explorar mais amplamente o espaço de soluções e adicionar uma componente mais aleatória ao processo. À medida que o algoritmo avança, a probabilidade de aceitação de soluções menos eficientes decresce com a temperatura, o que concentra a pesquisa numa parte mais específica do espaço de soluções [26, 80].

A Figura 8 apresenta o funcionamento do SA, com destaque para a diminuição, ao longo do tempo, da probabilidade de aceitação de uma solução que apresenta pior desempenho.

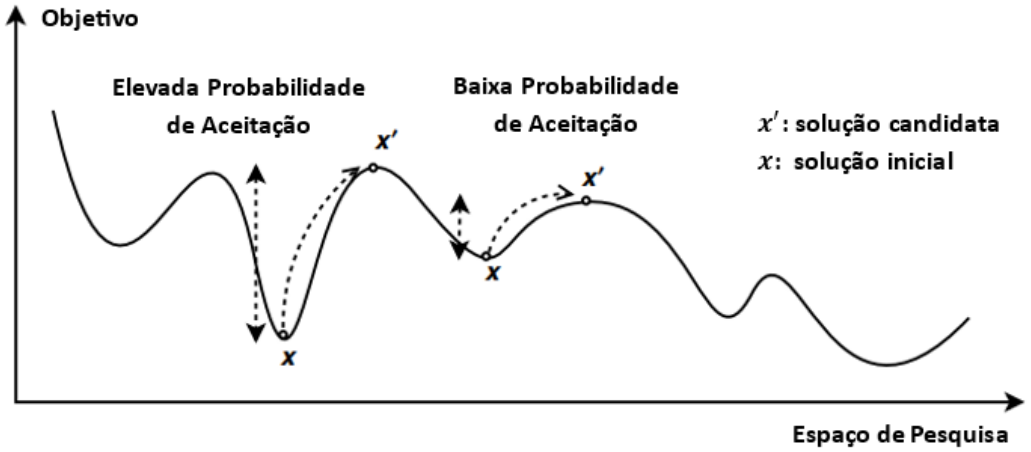


Figura 8- Estratégia de funcionamento do SA para evitar ótimos locais (adaptado de [26])

A eficiência do algoritmo é influenciada pelo Esquema (ou Programa) de Arrefecimento. Caso a diminuição da temperatura seja demasiado rápida, o algoritmo torna-se suscetível a ficar preso num ótimo local, mas se for uma diminuição muito lenta, o SA é capaz de apresentar melhores resultados, no entanto, é computacionalmente dispendioso [26]. O desafio consiste assim em definir o Esquema de Arrefecimento mais rápido capaz de garantir a convergência para o mínimo local [84].

Os parâmetros necessários para definir um Esquema de Arrefecimento são a Temperatura Inicial (TI), o Estado de Equilíbrio, a Função de Arrefecimento, e o Critério de Fim. Tendo em

consideração que o mais significativo é a Função de Arrefecimento, este será o único parâmetro abordado. Os restantes podem ser consultados em [26, 84].

A Função de Arrefecimento descreve a forma como o SA reduz a Temperatura T para o valor seguinte, ou seja, o modo como é atualizada. Na literatura existem disponíveis diversos modelos que representam o modo como esse valor de T é "atualizado", ou seja, como é que diminui, uma vez que a sua escolha depende essencialmente do problema em causa. Alguns dos mais comuns são apresentados de seguida, no entanto é importante mencionar que, como muitos estudos foram apresentadas à comunidade científica há muitos anos, vários autores têm utilizado diferentes nomenclaturas para Funções de Arrefecimento com o mesmo nome e propósito [84, 85].

→ Linear, introduzido por Kirkpatrick [78], onde o parâmetro T_k, que representa a Temperatura na iteração k, é calculado através da Equação (5). T_0 refere-se à Temperatura Inicial, e η é o fator de diminuição, situado normalmente entre 0 e 1;

$$T_k = T_0 - \eta \times k \tag{5}$$

→ Geométrico, representado pela Equação (6). Neste caso $\alpha \in]0,1[$ [26, 86].

$$T_k = \alpha \times T_0 \tag{6}$$

→ Diminuição muito Lenta, ou Não Linear, mencionado por Lundy e Mees [87]. No cálculo da Equação (7), β é um parâmetro pequeno positivo. Entre as várias propostas de cálculo deste parâmetro, pode-se enunciar a apresentada na Equação (8), onde M representa o número total de iterações;

$$T_{k+1} = \frac{T_k}{1 + \beta \times T_k} \tag{7}$$

$$\beta = \frac{T_0 - T_M}{(M-1) \times T_0 \times T_M} \tag{8}$$

→ Logarítmico, apresentado por Geman e Geman [88]. O parâmetro k representa o número de iterações, d é normalmente 1, e a constante C é a profundidade do mínimo global mais profundo, que é determinada empiricamente. Este programa de arrefecimento garante a convergência do SA para a solução ótima, no entanto a diminuição da temperatura é tão lenta que faz com apenas seja aplicado teoricamente.

$$T_k = \frac{C}{ln(d+k)} \tag{9}$$

Os Esquemas de Arrefecimento apresentados são estáticos, o que significa que são definidos completamente a priori. Talbi [26] e Triki et al. [85] mencionam que podem ser utilizados programas adaptativos, nos quais o cálculo do próximo valor da temperatura baseia-se no histórico de execução, isto é, depende da informação obtida durante a pesquisa. O objetivo é

manter o processo o mais curto possível, o que faz com que sejam frequentemente utilizados quando um pequeno número de iterações é efetuado a temperaturas elevadas, mas também pode ser usado com um grande número de iterações executadas a temperaturas baixas. Outros exemplos de modelos propostos ao longo do tempo que descrevem o modo como a Temperatura T decresce podem ser consultados nos estudos apresentados em [84, 85].

2.5.2. Tabu Search

Tabu Search é um Meta-Heurística introduzida em 1986 por Glover [89], que tem sido aplicada com sucesso em Problemas de Otimização Combinatória [90]. Destaca-se pela utilização de memória que armazena informação durante o processo de pesquisa com o objetivo de evitar que o algoritmo fique preso em soluções ótimas locais, direcionando assim a pesquisa para um espaço de pesquisa mais diversificado [91].

O algoritmo inicia-se com a obtenção de uma solução inicial, normalmente de forma aleatória. De seguida, procura e avalia, em cada iteração, novas soluções na vizinhança da solução atual, a escolhe aquela que apresentar melhor desempenho em relação à função objetivo. Ao contrário do que acontece nos algoritmos tradicionais de Pesquisa Local, o TS possui uma lista proibida, a Lista Tabu, que guarda movimentos ou soluções recentemente visitadas que não podem ser consideradas, o que significa que é necessário verificar se cada solução gerada não pertence a essa lista. Em cada pesquisa, a lista é atualizada com o movimento realizado, e o processo termina quando se verificar o critério de paragem estabelecido. Para evitar ótimos locais, o algoritmo permite "diminuir" o valor da função objetivo, aceitando assim soluções que apresentam desempenhos inferiores à solução atual [26, 92].

Para definir o processo de memória do TS são utilizados três conceitos: memória de curto prazo, memória de médio prazo, e memória de longo prazo.

O termo Memória de Curto Prazo, conhecido como Lista Tabu, refere-se ao armazenamento do histórico de soluções visitadas recentemente para impedir a existência de ciclos. Funciona como um desenho da trajetória de pesquisa e pode registar uma solução completa (por exemplo, 2 3 1 4), os índices dos algarismos trocados, e os movimentos realizados para obter uma nova solução atual (por exemplo $3 \rightarrow 1$). Este último tipo de registo é mais recorrente, uma vez que o seu tratamento é mais fácil e exige menos esforço computacional. Em cada iteração, a lista Tabu é atualizada, o que significa que o atual movimento é adicionado e o mais antigo é eliminado [93].

O desempenho do algoritmo é influenciado pelo tamanho da lista Tabu, que pode apresentar diferentes formas [26]:

- → Dimensão Estática. Observa-se quando a dimensão permanece constante. O valor associado pode depender das características do problema. Neste caso, listas pequenas concentram a pesquisa numa pequena área do espaço de soluções (intensificação), enquanto listas maiores exploram uma área mais ampla, promovendo a diversificação.
- → Dimensão Dinâmica. Significa que a dimensão varia ao longo do processo para diversificar e/ou intensificar a pesquisa.

→ Dimensão Adaptativa. Representa as situações onde o tamanho da lista é atualizado de acordo com a memória da pesquisa, como o desempenho na última iteração.

Caso o limite da lista Tabu seja k, o que significa que memoriza k soluções visitadas, o TS apenas é capaz de evitar um ciclo de tamanho no máximo k [26].

A Memória a Médio Prazo, relacionada com o conceito de Intensificação, consiste na exploração de informação das melhores soluções encontradas para orientar a pesquisa em regiões prometedoras do espaço de pesquisa. No exemplo mais comum, inicialmente definem-se os componentes associados a uma solução, o que é uma tarefa específica do problema, e, de seguida, a memória recente memoriza o número de iterações sucessivas que um componente esteve presente nas soluções visitadas [26, 94] .

A Memória de Longo Prazo é um mecanismo que pretende incentivar a Diversificação, forçando a pesquisa em áreas inexploradas do espaço de pesquisa [95]. A memória de frequência, utilizada frequentemente para representar este conceito, memoriza o número de vezes que cada "componente da solução" está presente nas soluções visitadas ou esteve envolvido nos movimentos selecionados [93].

Para caracterizar a conceção do TS, existem parâmetros que afetam o processo de pesquisa e a qualidade da solução do algoritmo, como os critérios de aspiração e os critérios de interrupção.

Os Critérios de Aspiração têm como objetivo impedir que o algoritmo evite movimentos potencialmente bons só porque estes estão temporariamente proibidos pela lista Tabu [96]. Um dos mais simples e comuns consiste em selecionar um movimento que está proibido, caso este permita alcançar um desempenho superior ao da melhor solução encontrada. Nestas situações, anular o efeito da proibição pode ser fundamental para evitar uma estagnação geral do processo de pesquisa [72].

Os Critérios de Interrupção ou Paragem são utilizados para interromper o algoritmo num determinado momento e obter assim a solução num período razoável. Gendreau et al. [93] referem que os critérios mais frequentes são após:

→ Um número fixo de iterações ou uma quantidade fixa de tempo de CPU *(Central Processing Unit)*;

→ Um certo número de iterações sem se observar uma melhoria no valor da função objetivo (o critério utilizado na maioria das implementações);

→ Quando o objetivo atinge um valor limite pré-especificado.

O algoritmo genérico que representa o TS é apresentado na Tabela 5. Neste caso s representa a solução atual, s_0 a solução inicial, $N(s)$ é o conjunto de soluções vizinhas de s, s' é a melhor solução vizinha de s, e $f(s')$ representa uma medida quantitativa da qualidade da solução s' em relação ao Problema de Otimização a ser resolvido (por exemplo, num problema de minimização de custos, representa o custo associado à solução s').

Tabela 5- Algoritmo representativo do *Tabu Search* (adaptado de [26, 93])

Gerar a solução inicial $s = s_0$
Inicializar a Lista Tabu
While Critério de Interrupção não Ocorrer **Do**
    Gerar as soluções vizinhas $N(s)$
    Selecionar a melhor solução vizinha, s'
    If $s' \in$ Lista Tabu Then
        If $f(s') =$ Critério de Aspiração Then
            $s = s'$
            Atualizar a Lista Tabu com s
        Else
            Remover melhor vizinho, s', de $N(s)$
        End
    Else
        $s = s'$
        Atualizar a Lista Tabu
    End
EndWhile
Output: Melhor Solução

O TS tem ampla aplicação em contextos relacionados a problemas de roteamento de veículos, mas também se destaca como uma estratégia particularmente eficaz em cenários que envolvem as diversas formas de implantações industriais. Recentemente, tem-se destacado notavelmente em Problemas de Escalonamento pela sua combinação com outros algoritmos, nos conhecidos algoritmos híbridos, que possibilitam a capitalização das vantagens inerentes a cada abordagem e superam algumas limitações intrínsecas a cada uma delas. Li et al. [90] propuseram um algoritmo que hibridiza os Algoritmos Genéticos e o TS para resolver o *Flexible Job-Shop Scheduling Problem* (Problema de Escalonamento de *Job Shop* Flexível) com o objetivo de minimizar o *Makespan*. O TS foi utilizado como uma estratégia para realizar a exploração, mas a principal vantagem é o facto do tamanho máximo de iterações ser ajustado de forma adaptativa durante o processo de evolução do algoritmo híbrido, o que possibilita um equilíbrio entre a diversificação e a intensificação, e economiza o tempo computacional. De maneira geral, a contribuição do TS reflete-se no desenvolvimento de um algoritmo capaz de obter os melhores resultados para os diversos problemas testados.

2.5.3. Outras Meta-Heurísticas

Desde a introdução das Meta-Heurísticas mais clássicas, entre o início dos anos 70 e o fim dos anos 90, o interesse pelo desenvolvimento de novos métodos foi aumentando, o que fez com que a proposta de novas Meta-Heurísticas tenha registado um crescimento exponencial [71]. Diversos estudos focados na identificação das mesmas foram desenvolvidos, como é o caso

apresentado em [73], onde são expostos detalhadamente cerca de 540 algoritmos Meta-Heurísticos descobertos e aplicados até o ano de 2022.

As duas Meta-Heurísticas abordadas anteriormente, SA e TS, são utilizadas com elevada frequência, mas existem outras que também merecem ser destacadas. No âmbito dos Algoritmos Evolutivos, que se inspiram na Teoria da Evolução de Darwin, é importante mencionar o *Genetic Algorithm*, pela sua ampla e notória utilização. É uma Meta-Heurística introduzida em 1972 por Holland [97], inspirada na seleção natural, que parte de uma população de solução candidatas que é evoluída através da aplicação de mecanismos de cruzamento, como seleção, crossover (recombinação genética) e mutação. Em cada passo, são privilegiadas as soluções mais promissoras [98]. Outras Meta-Heurísticas nesta categoria são a *Differential Evolution* (DE), introduzida por Price e Storn em 1995 [99], e *Gene Expression Programming*, proposta em 2001 por Ferreira [100].

Uma das categorias constituída por algoritmos amplamente aplicáveis é a Inteligência por Enxame. Rajwar et al. [73] demonstram que, até ao final de 2022, o *Particle Swarm Optimization* é o algoritmo mais referenciado. É uma Meta-Heurística proposta por Kennedy e Eberhart em 1995 [101], que se inspira no comportamento social de organismos coletivos, como bandos de pássaros ou cardumes de peixes. Inicialmente, um conjunto de partículas, cada uma representativa de uma solução candidata, é posicionado de forma aleatória num espaço de pesquisa. As partículas ajustam as suas posições e velocidades com base nas melhores soluções que conhecem e naquelas que são do conhecimento de todo o enxame. A melhor posição encontrada por cada partícula e a melhor posição global são continuamente atualizadas até satisfazer um determinado critério que finalize o processo [102, 103].

Neste campo também é possível mencionar o *Ant Colony Optimization* de Dorigo [104], o *Artificial Bee Colony* proposto por Karaboga em 2005 [105], e o *Grey Wolf Optimizer* e *o Whale Optimization Algorithm*, que são duas Meta-Heurísticas que, na última década, prosperaram rapidamente após serem propostas. O *Grey Wolf Optimizer*, proposto em 2014 por Mirjalili et al. [68], é inspirado na hierarquia social e no comportamento de caça dos lobos cinzentos. Para simular esta hierarquia de liderança, os lobos são classificados em quatro tipos: alfa (líder da alcateia), beta (transmite a informação do lobo alfa aos outros logo e ajuda-o na tomada de decisões), delta (encontra caminhos e protege os outros lobos) e ómega (obedece às regras dos lobos acima dele) [106]. Por sua vez, o *Whale Optimization Algorithm* foi introduzido em 2016 por Mirjalili e Lewis [107], e imita o mecanismo de caça das baleias jubarte no oceano.

A introdução de Meta-Heurísticas é um processo contínuo. Em 2023, Abder-Basset et al. [108] apresentaram o *Spider Wasp Optimizer*, um novo algoritmo que se baseia na replicação dos comportamentos de caça, nidificação e acasalamento das vespas-aranha fêmeas na natureza. Kaveh et al. [109] introduziram o *Orchad Algorithm*, inspirado na fruticultura, onde diversas ações como irrigação, adubação, poda e enxertia levam a um pomar onde a maioria das árvores cresce e produz frutos de forma adequada.

De facto, é notável a diversidade de Meta-Heurísticas existentes na literatura, o que reflete a complexidade e a adaptabilidade intrínseca aos Problemas de Otimização no mundo real, e sublinha a capacidade de inovação na procura por soluções eficientes. O utilizador dispõe assim

de uma ampla gama de alternativas para abordar os problemas das diferentes áreas, o que constitui um fator preponderante para alcançar o êxito.

Em [110], podem ser consultadas outras Meta-Heurísticas propostas recentemente.

2.6. Parametrização de Meta-Heurísticas

As Meta-Heurísticas são constituídas por um conjunto de parâmetros que desempenham um papel crucial na procura e seleção das melhores soluções. Apesar de serem técnicas apropriadas para resolver problemas complexos, frequentemente surge a dificuldade de identificar os parâmetros que direcionam a Meta-Heurística para a melhor solução, o que torna o processo de parametrização mais difícil do que a implementação da própria técnica [50, 111].

Formalmente, Hutter et al. [112] descrevem o processo de parametrização como: "Para uma Técnica de Otimização A, uma instância (ou conjunto de instâncias) do problema I, com um critério de desempenho C, o objetivo é definir os parâmetros de A que otimizam C em I". Por outras palavras, pretende-se identificar quais são os parâmetros que, numa determinada instância de um problema, permitem ao algoritmo obter o melhor desempenho.

Nos últimos anos, o interesse pela análise do desempenho de determinados parâmetros em diversas situações tem sido considerável, o que tornou este tema em algo muito utilizado pela indústria e alvo de estudo por parte da comunidade académica [113]. Esta é uma área de pesquisa considerada, no contexto das Meta-Heurísticas, como uma das mais desafiadoras, o que é justificável pela importância da mesma e pelo elevado impacto que uma definição de valores para parâmetros de um algoritmo de otimização pode ter [114].

Tendo em consideração as características do processo de parametrização e do próprio problema a resolver, a parametrização de uma técnica de otimização pode ser extremamente demorada [111]. Isto deve-se ao facto de, tipicamente, não se conhecerem as melhores soluções para o problema, e o número de combinações possíveis de parâmetros poder atingir dimensões elevadas, o que naturalmente aumenta a quantidade de experiências ou testes a realizar. A adicionar a isto, alguns autores, como Smit e Eiben [115], referem que alguns parâmetros são mais relevantes que outros, porque pequenas alterações nos seus valores podem afetar mais significativamente o desempenho do algoritmo, logo é necessário ser cuidadoso durante a definição dos seus valores.

De forma geral, a qualidade de uma configuração de parâmetros é dependente da instância do problema, o que significa que, para uma determinada Meta-Heurística, não é possível definir uma configuração de valores de parâmetros que vá apresentar o melhor desempenho para qualquer instância. Sendo assim, ajustar os parâmetros de uma Meta-Heurística é um ponto fundamental, porque é responsável pela eficiência (ou falta dela) do algoritmo. Parâmetros adequados ajudam o algoritmo a convergir para o ótimo mais rapidamente, o que melhora a sua eficiência. Atualmente, duas estratégias de parametrização significativamente utilizadas na literatura para ajustar os parâmetros deste tipo de algoritmo são *Offline* e *Online* [26, 116]. Enquanto a Parametrização *Offline* procede à seleção dos parâmetros antes da execução da Meta-Heurística, a Parametrização *Online* foca-se na alteração dos parâmetros dinamicamente,

ou seja, durante o processo de pesquisa do espaço de soluções. A Figura 9 apresenta algumas técnicas comuns que constituem cada um destes tipos de parametrização, baseando-se nos trabalhos apresentados por [12, 26, 117].

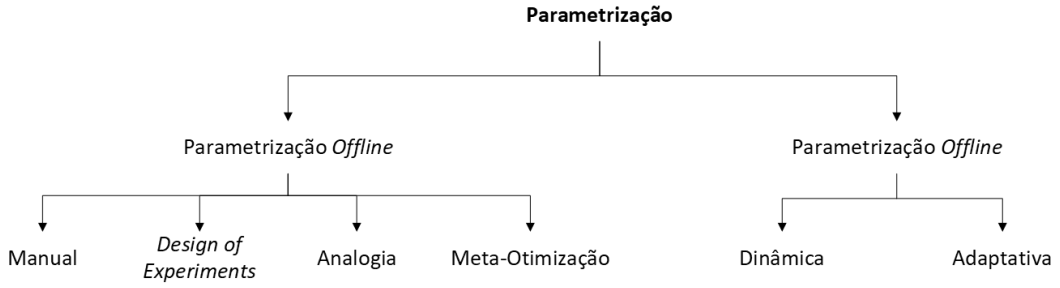


Figura 9- Tipos de Parametrização (adaptado de [12, 26, 117])

2.6.1. Parametrização *Offline*

O processo de Parametrização *Offline* determina, a priori, a configuração de parâmetros que deve ser usada na execução das Meta-Heurísticas, o que significa que os valores dos parâmetros são definidos antes da execução do algoritmo e não se alteram durante o processo [113]. Existe uma quantidade considerável de investigação sobre técnicas de Parametrização *Offline* aplicadas com sucesso em Meta-Heurísticas, o que tem como consequência a existência de diversas classificações para agrupar os diferentes métodos. Sendo assim, serão abordados alguns dos mais comuns, sendo importante destacar os Métodos de Parametrização Manual, a Parametrização por Analogia e a Parametrização por DOE *(Design of Experiments)*. Uma revisão precisa de outros métodos apresentados na Figura 9, como a Meta-Otimização, pode ser consultada, de forma mais geral, em [26, 118], enquanto em [12], o autor aborda, detalhadamente, diferentes técnicas que podem ser incluídas nesta classe de métodos.

Parametrização Manual

A Parametrização Manual é uma das abordagens mais tradicionais de seleção de parâmetros de um algoritmo [119, 120]. Nesta metodologia, o utilizador modifica, iterativamente, os parâmetros, com o intuito de melhorar o desempenho do algoritmo, baseando-se no conhecimento sobre o algoritmo e no problema a resolver.

De forma geral, o utilizador define os parâmetros iniciais e testa o algoritmo para perceber e avaliar o desempenho da Meta-Heurística. Posteriormente, procede a ajustes nos parâmetros, de forma individual, e executa o algoritmo novamente. Este procedimento é repetido, sempre com o objetivo de melhorar o desempenho ao algoritmo, até o utilizador estar satisfeito com o mesmo, o que lhe permite assim identificar os valores dos parâmetros que melhor se adaptam à resolução do problema em questão [121, 122].

Considerando os princípios fundamentais que estão na base da Parametrização Manual, pode-se considerar que é uma abordagem extremamente demorada e trabalhosa, não apenas devido ao processo em si, mas também porque depende da experiência do responsável pelo processo de parametrização do algoritmo. Neste caso, este processo requer a atenção especial de um

indivíduo particularmente qualificado porque, caso não esteja bem familiarizado com a Meta-Heurística, enfrentará dificuldades adicionais para selecionar e aplicar os parâmetros manualmente [113, 119]. Apesar de, ao longo de tempo, a configuração dos parâmetros ainda ser frequentemente realizada pelo projetista do algoritmo ou pelo utilizador final, devido às complicações associadas a este tipo de parametrização, tem-se aprofundado o estudo e desenvolvimento de métodos mais automáticos, capazes de ajudar os profissionais a identificar os conjuntos de parâmetros com melhor desempenho, a compreender e analisar a interação entre o desempenho do algoritmo, as instâncias do problema e os valores dos parâmetros selecionados, conduzindo assim a melhorias importantes no desempenho dos algoritmos para resolver diferentes Problemas de Otimização [123].

Parametrização por Analogia

A Parametrização por Analogia caracteriza-se por resolver um problema recorrendo às diretrizes estabelecidas em estudos publicados na literatura [122]. Neste caso, para resolver um determinado problema com uma Meta-Heurística, o utilizador pesquisa, na literatura, por implementações bem-sucedidas do algoritmo em problemas semelhantes, e replica os parâmetros utilizados por esses autores. Assim, em vez de parametrizar a Meta-Heurística, efetua-se uma pesquisa por parâmetros que já foram reportados como sendo capazes de permitir ao algoritmo obter uma boa solução [121, 124].

Naturalmente, desde a introdução dos primeiros algoritmos de otimização, como as Meta-Heurísticas, o estudo sobre a influência de cada parâmetro nos problemas mais comuns tem sido recorrente, o que cria, desde logo, uma base sólida para permitir a aplicação deste tipo de parametrização.

Design of Experiments (DOE)

Design of Experiments é uma abordagem estatística que envolve a realização/conceção de experiências para a modelação empírica de processos [125, 126]. Uma experiência é uma série de testes sistemáticos que procuram encontrar os fatores que têm o maior efeito numa variável de resposta [127]. Muitas vezes, é necessário examinar as influências de vários fatores, o que aumenta a complexidade doo processo de investigação. O DOE surge como uma estratégia que visa sistematizar o processo de execução destas experiências através da consideração de vários elementos: seleção cuidadosa dos fatores – que representam as variações dos parâmetros, seleção dos níveis – representativos dos diferentes valores dos parâmetros (ou seja, os intervalos de valores ou categorias), e definição da estrutura e do plano de execução das experiências para identificar a relação entre os fatores e a variável de resposta [128, 129].

A Parametrização por DOE procura minimizar a quantidade de experiências necessárias para uma análise, mantendo, ao mesmo tempo, resultados de elevada qualidade [130]. Esta abordagem é uma alternativa que não só permite obter conclusões sobre a influência de uma variável de entrada (como um parâmetro de uma Meta-Heurística) na saída do sistema, mas também analisar a interação entre múltiplas variáveis de entrada (os parâmetros). Em [127], os autores mencionam o *Factorial Design* e as Experiências de Taguchi como sendo duas das técnicas mais conhecidas de DOE.

No contexto das Meta-Heurísticas, a Parametrização por DOE pode ser utilizada no estudo do impacto dos parâmetros no desempenho de uma Meta-Heurística [122, 131]. Esta abordagem envolve a identificação de fatores – representada pela execução de uma série de corridas do algoritmo com diferentes combinações de parâmetros para determinar quais é que têm um impacto significativo no seu desempenho, selecionar os que apresentam melhor desempenho e utilizá-los então na Meta-Heurística para finalmente resolver uma determinada instância do problema [129]. Barbosa e Senne [132] empregam o DOE como uma ferramenta para definirem o espaço de procura de cada parâmetro do SA e de um *Genetic Algorithm* - o que corresponde à definição do intervalo de valores de cada um, para resolver o TSP. Em [133], Yu et al. utilizaram o *Factorial Design* para identificar os parâmetros estatisticamente significativos do *Genetic Algorithm* Por sua vez, Amoozegar e Rashedi [127] aplicam as Experiências de Taguchi para encontrar a melhor configuração dos parâmetros que caracterizam os *Gravitational Search Algorithms*, o que permitiu reduzir a complexidade do processo e conduzir o algoritmo ao seu melhor comportamento.

Embora seja um tipo de parametrização muito comum, esta abordagem apresenta uma desvantagem considerável, que é a exigência de um elevado custo computacional. Esta torna-se particularmente dispendiosa quando há um elevado número de parâmetros e os domínios dos respetivos valores são grandes, uma vez que é necessário realizar um número significativo de experiências [134].

Um tipo de métodos bastante comuns dentro deste tipo de parametrização são os Métodos *Racing*, O *F-Race* é uma técnica apresentada por Birattati et al. [135] e aprofundada em [136], é inspirada nas Técnicas de *Racing*, que foram propostas para a seleção de modelos em ML. O propósito deste método é selecionar as melhores configurações de parâmetros entre muitas possíveis através da aplicação de um procedimento de corrida que utiliza testes estatísticos sequenciais [137, 138]. Após um determinado número mínimo de instâncias (ou iterações) capaz de reunir informações para tomar uma decisão fundamentada, as configurações são avaliadas através do Teste de Friedman, que é aplicado para avaliar se existem diferenças estatisticamente significativas entre as configurações. Sempre que estiverem reunidas provas estatisticamente suficientes que evidenciem que um candidato apresenta um desempenho significativamente inferior aos outros, este é retirado, e avança-se para a iteração seguinte com os candidatos sobreviventes. Este processo continua até ser encontrada uma ou mais configurações com os melhores desempenhos num determinado conjunto de instâncias representativas de um problema [137, 139, 140].

Para uma visão mais detalhada sobre outros métodos de Parametrização *Offline*, é recomendada a leitura do artigo publicado por Huang et al. [124], no qual os autores abordam principalmente os métodos automáticos que apresentam maior aplicabilidade, como o REVAC (*Relevance Estimation and Value Calibration*), ParamILS, SPO (*Sequential Parameter Optimization*) e SMAC (*Sequential Model-based Algorithm Configuration*).

2.6.2. Parametrização *Online*

Os parâmetros dos algoritmos podem ser ajustados durante a execução dos mesmos [141]. Neste tipo de Parametrização, considerada *Online*, o controlo e a adaptação dos valores de cada parâmetro seguem normalmente dois passos: primeiro, é escolhido um conjunto inicial de valores de parâmetros, e, de seguida, é integrado um mecanismo de adaptação que altera os valores dos parâmetros mais relevantes, frequentemente baseado em informação específica que é armazenada durante a execução do processo [142].

O progresso do processo de pesquisa é monitorizado, com os valores a poderem ser ajustados de forma dinâmica (ou determinística) ou adaptativa, em tempo real, ou seja, durante a execução do algoritmo. Numa abordagem Dinâmica/Determinística, as modificações dos valores dos parâmetros são realizadas de forma aleatória ou determinística, sem considerar o processo de pesquisa. A regra de atualização de cada valor é determinada antes da execução do algoritmo. Nas abordagens Adaptativas, os valores dos parâmetros são ajustados em conformidade com o processo de pesquisa, através da utilização da memória ou de regras pré-estabelecidas. O objetivo é utilizar os vários dados disponíveis para tornar o processo de otimização mais eficiente [116, 117, 143]. A estas duas classes é normalmente adicionada a Auto-Adaptativa, no qual o processo de procura por bons valores para os parâmetros é realizado simultaneamente com a procura de soluções para o Problema de Otimização. Os parâmetros são codificados juntamente com outras variáveis, evoluem em conjunto durante o processo de otimização, e o próprio algoritmo é utilizado para resolver este problema [144].

Em [145], pode ser consultada uma revisão dos vários estudos focados na identificação dos diferentes tipos de Parametrização *Online* e nos algoritmos que se podem enquadrar em cada um. Por sua vez, em [146] é apresentada uma revisão sobre diferentes casos de aplicação, bem como diagramas das etapas que constituem os modelos de funcionamento de cada tipo. Neste caso, a maioria dos trabalhos retratam a aplicação principalmente em Algoritmos Evolutivos, que, devido às suas características, têm sido alvo de um interesse significativo por parte da comunidade científica. Em [147], o tamanho da população é atualizado a cada N avaliações de desempenho segundo uma estratégia de controlo de parâmetros Determinística. Em [148], numa estratégia Adaptativa, parâmetros de controlo como o tamanho da população, a recombinação e a mutação, são calculados durante o processo de otimização em função do progresso da pesquisa – sendo utilizados nomeadamente o melhor valor e o desvio padrão. Por fim, em [149], as taxas de mutação sofrem um processo de refinamento Auto-Adaptativo.

Devido à diversidade e qualidade das diferentes áreas, existem outras técnicas que têm sido aplicadas e podem ser incluídas num processo de Parametrização *Online*, nomeadamente aquelas relacionadas com a IA e o ML. No caso do RL, um tipo de Aprendizagem de ML, o seu uso tem como objetivo recorrer ao feedback da pesquisa para definir estados e ações que representam valores dos parâmetros [150]. De forma geral, como é apresentado em [151] com a aplicação de RL para otimizar o desempenho de um *Evolutionary Algorithm* (EA), os parâmetros são ajustados em cada episódio. Um estado reflete as principais prioridades em termos de parâmetros, num determinado estado é tomada uma ação, e são encontrados novos

valores dos parâmetros que são utilizados pelo EA. Em função do desempenho obtido, o sistema recebe um feedback e inicia-se um novo episódio.

No mesmo sentido de uma abordagem de aprendizagem sequencial, surge o uso de outras técnicas de ML, como técnicas de *Clustering*, no qual, de forma geral, o objetivo é adaptar os valores dos parâmetros de acordo com a informação reunida e extraída por *clusters*. Em [152], a aplicação do algoritmo *K-Means* permite agrupar a distribuição da população no espaço de pesquisa em cada iteração. Posteriormente, os valores são ajustados baseando-se no conhecimento que é extraído de cada *cluster*, nomeadamente nos seus tamanhos relativos.

Embora existam estudos promissores que mostram a capacidade de técnicas de ML de ajustar dinamicamente os parâmetros de forma eficaz, a quantidade de publicações robustas e aprofundadas ainda não é elevada e não apresenta a mesma dimensão quando comparada com a das técnicas de outros tipos de parametrização. Portanto, investir na pesquisa, tanto em termos de novas metodologias quanto na validação e experimentação das técnicas já existentes, posiciona-se como uma estratégia crucial para contribuir para o desenvolvimento e expansão desta área, criando assim uma base de conhecimento mais sólida e abrangente.

2.6.3. Parametrização do *Simulated Annealing*

O *Simulated Annealing* é uma das Meta-Heurísticas mais utilizada para resolver Problemas de Otimização Combinatória. A chave para o sucesso deste algoritmo inicia-se na escolha dos valores dos seus parâmetros, porque isso pode impactar significativamente o desempenho do algoritmo [153].

No caso do SA, o desafio é parametrizá-lo, de forma adequada, para maximizar o seu desempenho, isto é, obter as melhores soluções para um problema sem requerer demasiado tempo de processamento. De facto, este algoritmo não exige uma parametrização exaustiva, no entanto, quanto existem limitações comuns, como o tempo, torna-se crucial determinar os valores de cada parâmetro com precisão [50, 82]. Os parâmetros característicos do SA são: Temperatura Inicial, Esquema de Arrefecimento e *Epoch Length*. A adicionar a isto, existem outros parâmetros comuns a múltiplas Meta-Heurísticas que merecem uma atenção especial, como o Critério de Paragem ou Interrupção e a Estrutura de Vizinhança (EV).

Temperatura Inicial

Os métodos disponíveis para definir a TI podem envolver ou não informação relativa à instância do problema a resolver [154]. Tendo em consideração que a TI é utilizada para controlar a probabilidade de aceitação de soluções com pior desempenho que uma determinada solução, o seu valor inicial também é frequentemente determinado de modo a alcançar uma probabilidade de aceitação inicial desejada, o que envolve uma pesquisa aleatória pelo espaço de soluções. Kirkpatrick et al. [78] referem que o valor da Temperatura deve ser suficientemente elevado para que a probabilidade de aceitação seja próxima de 1, ou seja, as soluções sejam aceites na fase inicial do algoritmo, permitindo assim uma maior liberdade na exploração do espaço de pesquisa e evitando uma “aderência” ótimos locais. No entanto, um

valor elevado pode provocar um longo tempo de cálculo ou, em algumas situações, um desempenho indesejado [155].

Em [82], são realizadas várias transições, e o valor da Temperatura é escolhido de modo que a fração de transições ascendentes aceites seja igual a uma determinada probabilidade, $\mathrm{P_r}$ – Equação (10). Neste caso, Δ é o aumento médio dos valores da função objetivo calculado exclusivamente com transições ascendentes. Em [156], a métrica representada pela Equação (11) segue o mesmo princípio, mas Δ resulta da diferença entre o maior e menor valor da função objetivo obtidos e é recomendado um valor de $\mathrm{P_r}$ presente no intervalo $[0.7, 0.9]$. Em algumas situações, também pode ser usada a média da amostra, constituída pelas soluções visitadas.

$$T_0 = -\frac{\Delta}{ln\, P_r} \tag{10}$$

$$T_0 = -\frac{f^0_{máx} - f^0_{min}}{ln\, P_r} \tag{11}$$

Em [26], a diferença para a Equação (10) e a Equação (11) verifica-se no facto de que este estudo também considera o número de soluções que diminuem, ou seja, quando a transição é descendente. Neste caso, na Equação (12), $\mathrm{m_1}$ e $\mathrm{m_2}$ são os números de soluções que diminuem e aumentam a função objetivo nas experiências, respetivamente, e Δ^+ é a média dos valores da função objetivo aumentados. O rácio de aceitação, $\mathrm{a_0}$, à Temperatura $\mathrm{T_t}$, é definido como o número de transições ascendentes divido pelo número de transições efetuadas, e, para a fórmula apresentada, deve ter valores compreendidos entre 40% e 50%. Fórmulas similares com o mesmo princípio da apresentada na Equação (12) são abordadas em [79, 157].

$$T_0 = \frac{\Delta^+}{ln(m_1(a_0 - 1)/m_2 + a_0)} \tag{12}$$

Outros critérios que também realizam uma pesquisa aleatória pelo espaço de soluções são os apresentados por Burkard e Rendl [158] na Equação (13), no qual, tendo em consideração l soluções, os valores resultantes da função objetivo são tratados como uma série temporal. A partir daqui, para o cálculo da TI, a opção pode ser, por exemplo, multiplicar a constante k pela diferença máxima entre duas soluções candidatas consecutivas, $\Delta_{\mathrm{i,i+1}}$.

$$T_0 = k \times máx\left|\Delta_{i,i+1}\right| \tag{13}$$

Naturalmente, devido à diversidade de representações existentes, é fundamental perceber o tipo de problema a resolver e adaptar (ou, se necessário criar) os métodos existentes para o resolver da maneira mais eficaz. Outras variantes de todas as Equações apresentadas podem ser consultadas em [80, 82, 154, 159].

Esquema de Arrefecimento

O Esquema de Arrefecimento representa o modo como a Temperatura é diminuída ao longo da execução do SA [80]. A velocidade de convergência do algoritmo depende da Função de

Arrefecimento, e a sua influência tem sido investigada em vários trabalhos de investigação, como os apresentados em [160 – 162]. Para este parâmetro é recomendado optar por um Esquema que seja capaz de diminuir lentamente a Temperatura. Devido aos numerosos estudos que visam analisar a sua influência, na literatura existem diversas representações que descrevem a função de diminuição da Temperatura. Para o Esquema de Arrefecimento Geométrico, é recomendado em [79, 163] que o valor de α - C*ooling Rate*, seja entre $[0.50, 0.99]$, enquanto em o intervalo de valores é mais reduzido em [80, 164], onde se refere que valores entre $[0.80, 0.99]$ são capazes de permitir a execução do SA de uma forma eficiente porque diminuem a temperatura muito lentamente. Em [79], são apresentados procedimentos diferentes que calculam o *Cooling Rate* com base na Temperatura Inicial, na Temperatura Final e no número de patamares de Temperatura.

Para o Esquema de Arrefecimento apresentado por Lundy e Mees, conhecido como Linear e caracterizado pela constante β, que geralmente apresenta valores pequenos, Aarts e Var Laarhoven [157] aplicaram, com sucesso, duas taxas de resfriamento diferentes, $\beta = 0.5$ e $\beta = 0.9$, no entanto, como mencionado, também podem ser aplicados valores baixos, dependendo das características do problema [161]. Em [154], podem ser consultados outros Esquemas de Arrefecimento, como Logarítmico, Quadrático e Esquemas adaptados para determinados tipos de problemas, bem com os valores recomendados para os parâmetros que os caracterizam.

Epoch Length

O *Epoch Length*, L, corresponde ao número de iterações executadas até diminuir a Temperatura, ou seja, relaciona-se, ao mesmo tempo, com o número de soluções analisadas a uma determinada Temperatura [154].

Em [82, 165], é mencionado que o valor de L pode ser igual ao tamanho da instância do problema. Por exemplo, num Problema de Sequenciamento de Máquina Única, L pode ser representado pelo número de atividades existentes. Para além disso, também pode ser proporcional a esse tamanho, ou seja, a dimensão da instância n é multiplicada por um parâmetro constante k, o que faz com que $\mathrm{L} = \mathrm{k} \times \mathrm{n}$ [82, 166].

O valor de L também pode ser definido em função da Estrutura de Vizinhança. Neste caso, apresentado em [163], atualiza-se a Temperatura após um valor que resulta da multiplicação do número de vizinhos $(\mathrm{N_h})$ por uma constante, habitualmente definida pelo utilizador, ou seja, $\mathrm{L} = \mathrm{k} \times \mathrm{N_h}$. Numa versão mais adaptativa à medida que o SA é executado, em vez de se definir tamanhos fixos, o EL depende do desempenho da pesquisa. Abramson [167] atualiza a Temperatura após um certo número de soluções analisadas numa determinada iteração.

Em [154] podem ser consultadas outras representações do *Epoch Length* complementares.

Critério de Interrupção ou Paragem

O Critério de Interrupção ou Paragem do SA controla o momento em que a execução do algoritmo termina. Um critério comum, recomendado em [82], é terminar o SA quando se atinge um valor pré-determinado de iterações. Em [53, 168], os autores mencionam que a procura no espaço de soluções também pode ser encerrada quando se atinge um determinado

valor de Temperatura Mínima, representado como $\mathrm{T}_{\mathrm{min}}$. Este valor pode ser determinado com base no esquema de TI implementado, ou ser fornecido pelo utilizador. Outro método comum mencionado em [82], é restringir o tempo de execução através de um tempo máximo de CPU.

Numa vertente mais adaptativa, em [169] aborda-se a implementação de critérios baseados na observação da execução real do algoritmo. Neste caso, a execução do algoritmo é interrompida após um determinado número de soluções candidatas não serem aceites sucessivamente. Em [170], num trabalho que estuda a aplicação do SA para minimizar globalmente funções constituídas por muitas variáveis contínuas, os autores implementam vários critérios de paragem mencionados nesta revisão.

2.7. Machine Learning

Machine Learning é um domínio da IA dedicado à conceção de técnicas e algoritmos que permitem que os sistemas computacionais sejam capazes de processar, aprender e extrair informação significativa a partir dos dados [171]. O foco é detetar automaticamente padrões nos dados e, de seguida, utilizar os padrões descobertos para prever dados futuros ou para tomar outros tipos de decisões em condições de incerteza [172].

A história do ML inicia-se na década de 1940, quando surgiram as primeiras tentativas de criar sistemas capazes de imitar o comportamento humano. Em 1943, Pitts e McCulloch [173] deram um passo significativo com o desenvolvimento do primeiro modelo de redes neuronais, e mais tarde, em 1950, Turing [174] apresentou o Jogo da Imitação, onde introduziu a questão "As máquinas podem pensar?", o que tornou a sua publicação uma das mais emblemáticas na área da IA porque fez todos pensar no que aconteceria se a máquina assumisse o papel de um Humano, isto é, se ela iria agir da mesma maneira que uma pessoa age [175].

Em 1959, num artigo publicado no *IBM Journal*, Arthur Lee Samuel introduzir o termo *Machine Learning* pela primeira vez, quando apresentou o seu programa *Game of Checkers*. O investigador americano utilizou os registos de exemplos de boas jogadas praticadas pelos jogadores profissionais para programar o computador, e, durante a explicação do jogo, abordou diversos conceitos que permitiram definir o que é o ML [176]:

> "*Machine Learning* é o campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados" – Arthur Samuel, 1959 [176]

Nos anos subsequentes, foram apresentados diversos trabalhos na área, evidenciado assim um crescimento notável. Contudo, foi a partir do início da década de 1990 que o ML começou a se destacar como uma das áreas mais bem-sucedidas dentro da IA devido ao desenvolvimento de programas capazes de desafiar os jogadores profissionais [175].

Mitchell, em 1997 [177], utilizou uma definição mais orientada à engenharia, mas que se tornou popular pela simplicidade com que o mesmo aborda a questão do ML. Este autor estabelece que "um programa pode ser classificado como estando a aprender com a experiência E, relacionada a uma classe de tarefas T com um indicador de desempenho D, se o seu desempenho na tarefa T, medida por D, melhora com a experiência E". Esta abordagem de Mitchell pode ser traduzida para os diversos exemplos onde o ML é aplicado, como a

identificação dos dígitos (0 a 9) presentes numa imagem. Neste caso, T seria a classificação de dígitos manuscritos, E seria o conjunto de dados de treino que consiste em várias imagens de dígitos manuscritos previamente rotulados, e D o rácio de números classificados corretamente.

Nas últimas duas décadas, o ML progrediu drasticamente e consolidou-se como uma ferramenta indispensável em diversas áreas. A sua aplicação abrange o diagnóstico médico, o controlo robótico, o desenvolvimento de software prático para visão computacional, o processo de linguagem natural, a análise de dados no mercado financeiro, entre outras áreas. A comunidade científica reconhece que, em várias aplicações, é mais fácil treinar um sistema mostrando-lhe exemplos das entradas e das saídas, em vez de o programar manualmente [178]. Alguns dos recentes desenvolvimentos que envolvem ML nos seus projetos são a *Baidu's Deep Voice*, os modelos OpenAI GPT e o Robot Spot lançado pela Boston Dynamics.

2.8. Tipos de Aprendizagem

O campo da IA e do ML é vasto e dinâmico, e abrange várias abordagens destinadas a capacitar algoritmos a aprenderem a partir de dados e realizar previsões. A diversidade de técnicas de aprendizagem está intrinsecamente ligada a diversos fatores, incluindo o modo como os algoritmos assimilam informações, o tipo de dados disponíveis para os treinar e o resultado desejado. Sendo assim, os algoritmos podem ser categorizados em três grupos: Aprendizagem Supervisionada, Aprendizagem Não Supervisionada e Aprendizagem por Reforço (RL) [179, 180].

Esta Secção 2.8 será focada nos três tipos de aprendizagem mencionados, no entanto, é importante destacar a existência de outros, como a Aprendizagem Semi-Supervisionada, a Aprendizagem Auto-Supervisionada e a Aprendizagem em Várias Instâncias [181].

2.8.1. Aprendizagem Supervisionada

Na Aprendizagem Supervisionada, um algoritmo é treinado utilizando um conjunto de dados rotulados, o que significa que lhe são apresentados os dados de entrada emparelhados com os rótulos de saída desejados correspondentes. O objetivo consiste em fazer um mapeamento das entradas para as saídas com base nos exemplos fornecidos durante o teste [178, 182].

A entrada de treino, X_i, pode ser um vetor clássico, ou objetos mais complexos, como imagens, documentos, uma série temporal, uma fórmula molecular ou gráficos. A saída, Y_i, assume a configuração de uma variável categórica ou nominal para um problema de classificação, ou um valor real em situações de problemas de regressão [175]. Em ambos os parâmetros (entrada e saída), são necessários agentes, normalmente um ser humano, para fornecer a informação.

Após o modelo ter sido treinado com êxito, pode ser utilizado para prever um resultado específico de interesse com base em dados novos ou não vistos [183].

Duas tarefas típicas associadas à Aprendizagem Supervisionada são a classificação e a regressão, que se diferenciam na natureza da variável de saída. A classificação concentra-se na atribuição de uma categoria ou um rótulo aos dados de entrada, e a saída é um valor discreto

que representa uma associação a uma classe, como acontece com a deteção de spam ou a identificação de objetos numa imagem. Por sua vez, a regressão foca-se na previsão de valores numéricos, o que significa que a saída é de natureza contínua (por exemplo, o preço de um imóvel com base em características do mesmo) [184].

A Figura 10 apresenta exemplos reais ilustrativos da diferença entre classificação e regressão.

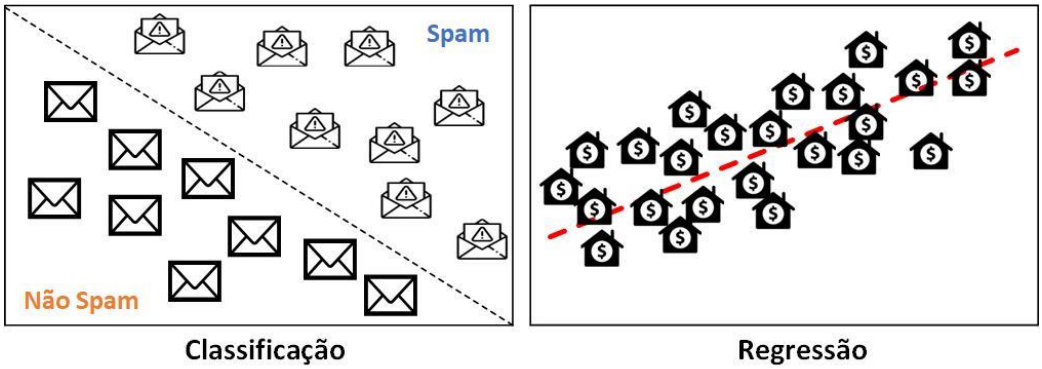


Figura 10- Classificação versus Regressão na Aprendizagem Supervisionada

A Aprendizagem Supervisionada inclui diversos algoritmos, sendo alguns dos mais utilizados os seguintes: *Linear Regression, Decision Tree, K-Nearest Neighbors, Neural Networks, Random Forest* e *Support Vector Machine* (SVM). É importante destacar que alguns algoritmos de regressão podem ser utilizados para classificação, e vice-versa [185].

A versatilidade inerente ao processo de Aprendizagem Supervisionada permite aplicá-la em vários domínios, como a análise preditiva (previsão temporal, demográfica e financeira), a deteção de fraudes, os diagnósticos médicos (previsão de doenças com base em dados médicos), os sistemas de recomendação, o reconhecimento de imagens e objetos, entre outros.

2.8.2. Aprendizagem Não Supervisionada

A Aprendizagem Não Supervisionada utiliza dados desprovidos de quaisquer rótulos ou especificações prévias. Estes dados de treino não possuem uma variável de saída fixa, o que indica a ausência de um supervisor que forneça informações adicionais, tais como respostas corretas ou o grau de erro associado a cada observação. O propósito primordial é habilitar o sistema a identificar padrões ocultos e interessantes nos dados de entrada [175, 186].

De acordo com Géron [185], neste campo específico de Aprendizagem podem ser identificadas diversas tarefas, sendo importante destacar três delas:

- → *Clustering*. Consiste no agrupamento, em *clusters*, de elementos que partilham propriedades comuns;
- → Redução da Dimensionalidade. Refere-se ao processo de redução do número de características ou variáveis num conjunto de dados, preservando o máximo de informação relevante possível. O objetivo é simplificar o conjunto de dados, melhorando assim a eficiência computacional;

→ Deteção de Anomalias (ou Deteção de *Outliers*). Envolve a identificação de instâncias num conjunto de dados que se desviam significativamente dos padrões.

Alguns dos algoritmos de Aprendizagem Não Supervisionada mais utilizados são o *K-Means Clustering*, o *Density-Based Spatial Clustering of Applications with Noise*, o *Principal Component Analysis* e o *AutoEncoder*.

Devido à capacidade de compreender e detetar padrões complexos num conjunto de dados, a Aprendizagem Não Supervisionada desempenha um papel fundamental em diversas disciplinas. Isto inclui o marketing e a segmentação de clientes, onde, por exemplo, identifica-se grupos de clientes com comportamentos similares, a área da saúde com a interpretação de imagens médicas, o mercado financeiro através da previsão de tendências, e a análise de redes sociais para entender a dinâmica das interações online.

2.8.3. Aprendizagem por Reforço *(Reinforcement Learning)*

A essência do RL reside no facto de o sistema aprender que ações deve tomar numa situação, de modo a maximizar um sinal de recompensa. Ao contrário de outros tipos de Aprendizagem, existe um Agente que não recebe informação sobre a ação a executar ou o resultado ideal, mas este é incumbido de descobrir sozinho quais ações resultam em maiores recompensas através de um processo experimental (mediante tentativa e erro) [187].

O RL é uma ferramenta poderosa para treinar modelos de IA que ajudam a aumentar a automatização ou a otimizar a eficiência operacional de sistemas sofisticados, como a robótica através do treino de robots capazes de executar várias tarefas, a área das finanças para desenvolver sistemas capazes de tomar decisões de transações nos mercados financeiros, a área da saúde com a identificação dos tratamentos adequados para os doentes em cada situação, em jogos para desenvolver agentes para os jogar, entre outros [188].

Qualquer problema de RL tem dois componentes principais: o Agente e o Ambiente. O Agente, que necessita de ter um objetivo bem definido, é a entidade que decide as ações a tomar e que é capaz de averiguar o estado do Ambiente, mesmo que com incertezas. O Ambiente é onde o Agente opera e relaciona-se com o problema a resolver. Sutton e Barto [189] referem mais quatro componentes que são considerados críticos para qualquer sistema: a recompensa, a política, a função valor de cada estado e o modelo do Ambiente.

O processo de interação entre o Agente e o Ambiente é representado por um processo de decisão de Markov, que é uma evolução das cadeias de Markov. Na abordagem apresentada na Figura 11, a interação acontece em instantes de tempo discretos ($t = 0,1,2, \ldots$) por motivos de simplificação, uma vez que, na prática, as observações são contínuas. Em cada instante, um Agente recebe uma representação do estado do Ambiente, $s_t \in S$, e com base nela, seleciona uma ação, $a_t \in A(s)$. No instante seguinte, como consequência da sua ação, o Agente recebe uma recompensa numérica, $R_{t+1} \in R$, e encontra-se num novo estado, S_{t+1}. Este procedimento é repetido, e cada iteração é designada por episódio. Este processo descrito permite assim criar a sequência (ou trajetória) apresentada na Equação (14) [189, 190].

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \quad (14)$$

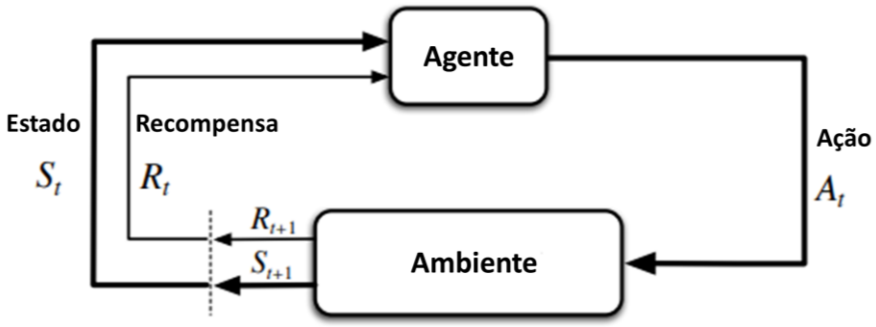


Figura 11- Relação entre Agente e Ambiente (adaptado de [189])

Num processo de decisão de Markov, o conjunto de ações, estados e recompensas é definido por $(\mathrm{S}, \mathrm{A}, \mathrm{R})$ e inclui sempre um número finito de elementos. Os valores de $\mathrm{R_t}$ e $\mathrm{S_t}$ dependem apenas do estado anterior $\mathrm{S_{t-1}}$; i.e., num determinado momento de tempo t, a probabilidade de se verificar o estado s e a recompensa r (em que $\mathrm{s}, \mathrm{s}' \in \mathrm{S},\ \mathrm{r} \in \mathrm{R}$ e $\mathrm{a} \in \mathrm{A(s)}$), e pode ser definida pela Equação (15) [189, 191]:

$$p(s', r|s, a) = Prob(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a) \quad (15)$$

O modelo do Ambiente procura reproduzir o comportamento do Ambiente, de forma que se possam efetuar inferências e prever como este irá reagir a uma ação. Ao ser fornecido um estado e uma ação, o modelo deverá retornar a recompensa associada a essa ação e calcular o próximo estado do Ambiente. Isto significa que o modelo assume um papel importante porque, com base em expectativas do que acontecerá em situações futuras, este permite tomar decisões sobre uma ação antes dela ser executada [189, 192].

Recompensa

A recompensa é a forma como o objetivo do Agente é definido, e define o quão positivo é (ou não) executar uma ação num contexto específico [193]. Como o Agente não sabe a priori qual é a melhor ação, e como as ações podem influenciar não apenas a recompensa imediata, mas também as recompensas subsequentes, este tem de analisar as ações que maximizam a soma total das recompensas ao longo do tempo. A soma total das recompensas, $\mathrm{R_t}$, pode ser calculada de acordo com a Equação (16), na qual t é um determinado instante de tempo e T é o instante de tempo final [186, 194].

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T \quad (16)$$

Nas situações onde o T é ∞, a soma total das recompensas seria infinita. Para solucionar isto, usa-se normalmente um fator de desconto que ajusta a importância das recompensas ao longo do tempo. O Agente tem assim o objetivo de maximizar a recompensa total descontada de acordo com a Equação (17) [189]. Neste caso $\mathrm{r_{t+n}}$ representa a recompensa no instante $\mathrm{t+n}$ e γ o fator de desconto.

$$R_t = r_{t+1} + \gamma \times r_{t+2} + \gamma^2 \times r_{t+3} + \ldots = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \quad (17)$$

O fator de desconto, γ, pode variar entre $0 \leq \gamma \leq 1$, e permite modelar o comportamento do Agente em função do seu objetivo, neste caso a curto ou longo prazo. Caso apresente o valor 0, a recompensa imediata é a mais relevante. À medida que o seu valor aumenta, o Agente dá mais importância às recompensas futuras, o que significa que quando é 1, o Agente avalia cada ação com base na soma total das suas recompensas futuras [195, 196].

Em cada momento, a informação que o Agente recebe do Ambiente refere-se exclusivamente à recompensa imediata associada à ação que acabou de ser executada. Nesse sentido, o Agente depara-se com o dilema de escolher entre explorar estados e ações desconhecidos para adquirir informações adicionais sobre o ambiente e as recompensas, e explorar as informações já recolhidas para otimizar a sua recompensa, o que é conhecido como o compromisso entre *exploration* (exploração) e *exploitation* (aproveitamento) inerente ao RL [179, 197].

Política e Funções de Valor

A política, representada por π e conhecida como uma componente central de RL, estabelece o modo como o Agente se comporta num determinado momento, através do mapeamento dos estados do ambiente para as ações que o Agente deve ter quando se encontra nesses estados [187, 198]. Pode ser determinística, definida pelo facto de, para cada estado, o Agente selecionar uma ação específica, o que significa que se mapeia cada estado e cada uma das ações disponíveis - $\pi(s_t) = a_t$, em que $s_t \subset S$, $a_t \subset A$, ou estocástica, quando, para além dos estados, também se mapeiam as soluções mas associadas a uma probabilidade - $\pi(a_t|s_t) = p_i$, em que $s_t \subset S$, $a_t \subset A$, $0 \leq p_i \leq 1$ [199, 200].

Dado um estado, uma política devolve uma ação a executar. O objetivo do Agente é aprender uma política que maximize o retorno esperado no ambiente [201]. Sendo assim, o Agente deve optar por ações que conduzem a estados com maior valor e não com a maior recompensa porque o seu objetivo é acumular a maior recompensa total a longo prazo [192, 197].

Para se definir políticas, é necessário estimar funções de valor, que permitem ao Agente ter uma visão a longo prazo. Uma função valor de estado avalia o quão bom é estar num determinado estado, considerando as recompensas futuras que o Agente pode esperar seguindo uma determinada política fixada [195, 202]. Dado um estado, $s \in S$, e considerando a política π, a função de valor compreende assim uma soma de recompensas descontadas, tal como é apresentado na Equação (18). Neste caso E_π corresponde ao valor esperado quando as ações são escolhidas segundo a política π [189].

$$V_\pi(s) = E_\pi[R_r|s_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k \times r_{t+k+1}|s_t = s\right] \quad (18)$$

Para avaliar a qualidade da aplicação de uma ação, $a \in A(s)$, a um estado específico, $s \in S$, considerando a política π, utiliza-se normalmente uma função de qualidade, Q_π. Para uma determinada política, a função Q pode ser calculada através da Equação (19) [189, 203].

$$Q_\pi(s,a) = E_\pi[R_r|s_t = s, a_t = a\ ] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k \times r_{t+k+1}|s_t = s, a_t = a\right] \quad (19)$$

Uma política que maximiza o retorno esperado ao escolher uma determinada ação é assim a que apresenta o maior valor de Q, como apresenta a Equação (20) [189].

$$\pi(s) = arg\ max Q(s,a)\ , \forall_{s \in S} \quad (20)$$

Qualquer algoritmo de RL tem o objetivo de, em cada iteração, melhorar a sua política até alcançar a política ótima, representada na literatura por π^*, que naturalmente se caracteriza por ser melhor que as restantes. Para cada política ótima, é possível também representar a função de valor ótima, isto é, o que permite avaliar a qualidade de um estado, que é $V^*(s)$, e a função de qualidade de uma ação, representada por $Q^*(s,a)$.

Determinar o valor de Q^* significa saber qual é a ação que o Agente deve tomar, no entanto, devido às particularidades e características dos problemas reais, nem sempre é possível de calcular. Em vários casos, é necessário prever todos os casos possíveis, a probabilidade de acontecerem e até mesmo a recompensa total associada a cada um, o que pode significar um trabalho exaustivo e um esforço computacional elevado [191].

Tendo isto em consideração, existe uma extensa variedade de algoritmos de RL que tentam estimar os valores das funções de valor. Estes algoritmos são normalmente divididos em três classes: abordagens baseadas em Programação Dinâmica, métodos de Monte Carlo e métodos de Diferença Temporal [189, 197]. Como a quantidade de parâmetros associados ao RL é elevada, também existem outras classificações populares, como *model-based vs model-free*, que se diferenciam pela capacidade de construir um modelo interno do ambiente ou aprender diretamente com a experiência [204], ou até *off-policy vs on-policy*, que se referem ao modo como as políticas são avaliadas e atualizadas durante o processo de aprendizagem [205].

Um dos algoritmos que se destacam no contexto de RL é o *Q-Learning,* que é capaz de interagir com o Ambiente para aumentar o seu conhecimento e consequentemente melhorar a qualidade das decisões tomadas. Sendo assim, este algoritmo será abordado detalhadamente de seguida, no entanto outros algoritmos importantes que merecem ser destacados são os *Policy Gradient Methods, o Deep Q-Network* e o *Actor-Critic* [206].

Algoritmo *Q-Learning*

O *Q-Learning* é um dos algoritmos de RL mais utilizados devido à sua simplicidade. É usado para aprender políticas ótimas em processos de decisão de Markov [207]. Introduzido por Watkins [208, 209], é um algoritmo sem modelo *(model-free)* porque o Agente interage diretamente com o Ambiente e aprende com a experiência. Isto significa que não recorre à distribuição de transição e a funções de recompensa para obter a política ideal e criar o respetivo modelo, nem

"simula" possíveis estados e recompensas futuras para tomar decisões, como fazem os algoritmos baseados em modelo [195, 210].

É um método baseado em valor *(value-based)*, o que significa que treina a função de valor para aprender qual é o estado mais valioso e como deve agir, o que é diferente dos algoritmos baseados em políticas que treinam a política diretamente para aprender que ação devem tomar num determinado estado [211]. Também se inclui na categoria de *off-policy* porque avalia e atualiza uma política que difere da política usada para executar uma ação [195, 204].

O algoritmo *Q-Learning* utiliza uma tabela (tabela Q) para armazenar os valores Q, conhecidos como valores de estado-ação. Estes valores estimam a recompensa cumulativa esperada que o Agente receberá se tomar a ação a no estado s e seguir a política ótima a partir desse momento [212]. A cada par estado-ação $(s, a) \in S \times A$ é assim associado um valor $Q(s, a) \in R$, que é atualizado sempre que uma ação é tomada, de acordo com a Equação (21). Neste caso s é o estado atual, s' é o estado seguinte após a realização da ação a, e a' é uma ação possível no estado s' . O α é a taxa de aprendizagem ($0 < \alpha \leq 1$), o r representa a recompensa recebida após a execução da ação a, e γ é o fator de desconto [189, 208].

$$Q(s,a) \longleftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \tag{21}$$

Dois parâmetros presentes na Equação (21) requerem uma atenção especial. A taxa de aprendizagem, α, regula a rapidez com que os valores da tabela Q são alterados. Neste caso, para taxas de aprendizagem mais baixas a convergência de aprendizagem é mais lenta, mas estável. Normalmente inicia-se o processo com valores de α mais elevados para permitir alterações mais rápidas, e depois reduz-se gradualmente [213]. O segundo, γ, que é o fator de desconto ($0 < \gamma \leq 1$), é responsável por determinar a influência das recompensas futuras. Como referido anteriormente nesta Secção 2.8.3, se o valor for 0 o Agente apenas se preocupa com a recompensa atual, enquanto se for 1 significa que também considera todas as recompensas futuras [195].

A Tabela 6 apresenta o algoritmo genérico do *Q-Learning*.

Tabela 6- Algoritmo representativo do *Q-Learning* [208]

Dados de Entrada:
$Q(s, a)$ com valores atribuídos, $\forall_{s \in S}, \forall_{a \in A(s)}$
Política π a considerar
Do
  Inicializar o estado s
  While não atingir um estado s terminal **Do**
    Escolher uma ação a do estado s utilizando a política π
    Aplicar a ação a, observar a recompensa r e o estado seguinte s'
    Atualizar o valor-Q: $Q(s, a) \longleftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \longleftarrow s'$
  EndWhile
Until não existirem episódios

Um aspeto importante relacionado com o algoritmo *Q-Learning* é a necessidade de, no processo de aprendizagem, alcançar um equilíbrio entre os conceitos de *exploration* e *exploitation*, ou seja, exploração e aproveitamento, porque isso melhora significativamente o desempenho do Agente [214, 215]. Quando se abordam estes conceitos, surge um dilema relativo à estratégia a aplicar, que tem sido estudado cada vez mais dentro da comunidade científica: É mais benéfico explorar o conhecimento já adquirido, seguir um caminho conhecido e utilizar as ações conhecidas porque apresenta elevada recompensa *(exploitation),* ou deve-se explorar novas ações para descobrir novas recompensas, o que significa explorar novos estados para encontrar novas e melhores políticas *(exploration)*? De facto, limitar-se a *exploitation* – aproveitamento, pode impedir o alcance de pontuações elevadas e não melhorar as ações, enquanto o foco apenas na *exploration* – exploração, pode ter como consequência o facto de o Agente ficar preso na sua política atual sem analisar as trajetórias possíveis [195, 216].

Entre as estratégias mais comuns insere-se a estratégia de seleção *epsilon-greedy*. Dada uma função $\mathrm{Q}(\mathrm{s},\mathrm{a})$, o Agente seleciona a melhor ação, isto é, aquela que maximiza um valor de Q para um determinado estado, com uma probabilidade de $(1-\epsilon)$. Com uma probabilidade ϵ é escolhida aleatoriamente uma ação do conjunto de ações $(\mathrm{a} \in \mathrm{A})$. A Equação (22) representa a abordagem da estratégia mencionada [189, 195].

$$a = \begin{cases} arg\,max\,Q(s,a), & com\ probabilidade\ 1-\varepsilon \\ qualquer\ ação\ selecionada\ uniforme\ e\ aletoriamente\ em\ A, & com\ probabilidade\ \varepsilon \end{cases} \quad (22)$$

O valor de ε, que pode ser entre 0 e 1, influencia o desempenho do algoritmo durante o processo de aprendizagem. Neste caso, valores elevados conduzem a uma estratégia de exploração, o que significa reduzir a probabilidade de agir de acordo com o ótimo. Por sua vez, valores baixos têm tendência a fazer com que o Agente explore mais ações e aumenta a probabilidade de encontrar a política ótima. Sendo assim, a tendência é que o valor diminua ao longo do treino para permitir ao Agente comportar-se progressivamente de forma mais autónoma. De qualquer modo, independentemente da forma como a ação foi escolhida, o valor de Q na tabela é sempre atualizado [195, 214].

O *Q-Learning* torna-se desvantajoso em situações onde o número de estados e ações são significativamente elevados. Como os valores de Q são armazenados numa tabela, a quantidade de memória para guardar e atualizar a tabela, bem como o tempo necessário para explorar cada estado, aumenta consideravelmente, o que faz com que, em muitos casos, sejam utilizados outros algoritmos de ML para solucionar o problema [217, 218].

Um exemplo simples e conhecido da aplicação do algoritmo *Q-Learning* é um problema de RL aplicado a um ambiente em grelha, conhecido como *"GridWorld Problem"*. Neste tipo de problemas, representado na Figura 12, o Agente navega numa grelha 5x5 e tem quatro ações possíveis: cima, baixo, esquerda e direita. O estado inicial do Agente é um dos cinco quadrados da grelha na parte inferior, selecionado aleatoriamente. Os quadrados com o ouro (0,3) e a bomba (1,3) são estados terminais, o que significa que se o Agente se encontrar num destes quadrados, o episódio termina, e depois o novo episódio inicia-se com o Agente no estado inicial. A função de recompensa é -1 por cada ação, $+10$ adicionais por encontrar o ouro, e

-10 adicionais se atingir a bomba, ou seja, por exemplo, quando alcançar o quadrado do ouro a recompensa é $-1 + 10 = 9$.

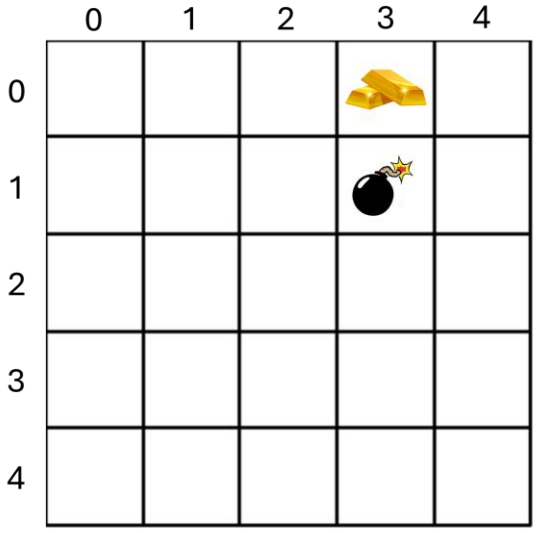


Figura 12- Representação do problema *"GridWorld"*

O objetivo do problema é utilizar o *Q-Learning* para calcular a melhor política para encontrar o ouro com o menor número de passos possíveis. Assim, para perceber o papel e a capacidade do Agente, este problema foi resolvido através de dois procedimentos: recorrendo a um Agente Aleatório que, em todos os momentos, apenas seleciona uma ação aleatória sem nenhum fundamento, e utilizando um Agente construído com a implementação do *Q-Learning*. O código completo com comentários explicativos de cada parte está presente no Apêndice A.

A Figura 13 e a Figura 14 apresentam as curvas de aprendizagem de um Agente Aleatório e um Agente *Q-Learning*, respetivamente, após a execução de 500 episódios, com o Agente a efetuar, no máximo, 1000 passos por episódio, ou com o episódio a terminar caso se mova para a célula que contém o ouro. Neste caso, para o Agente Aleatório, observa-se que a recompensa acumulada varia significativamente ao longo das iterações. Como expectável, este Agente não apresenta um padrão de melhoria do seu desempenho, a curva de aprendizagem evidencia uma grande dispersão nos valores, o que confirma a ideia de que a ausência de um mecanismo de aprendizagem impediu que ele fosse capaz de adaptar as suas ações de acordo com as experiências passadas. Por outro lado, o Agente *Q-Learning* apresenta um comportamento significativamente diferente. Inicialmente a recompensa acumulada varia porque o Agente ainda está a explorar o ambiente e a construir a sua tabela, no entanto, à medida que o número de iterações aumenta, existe, primeiro, uma tendência clara de aumento da recompensa e, posteriormente, uma estabilização e convergência desse valor. Este comportamento confirma o efeito de aprendizagem do Agente e a eficácia do algoritmo *Q-Learning* em aprender e otimizar a política de ações através da interação com o Ambiente e da atualização contínua da tabela de valores Q com base nas recompensas recebidas em cada momento.

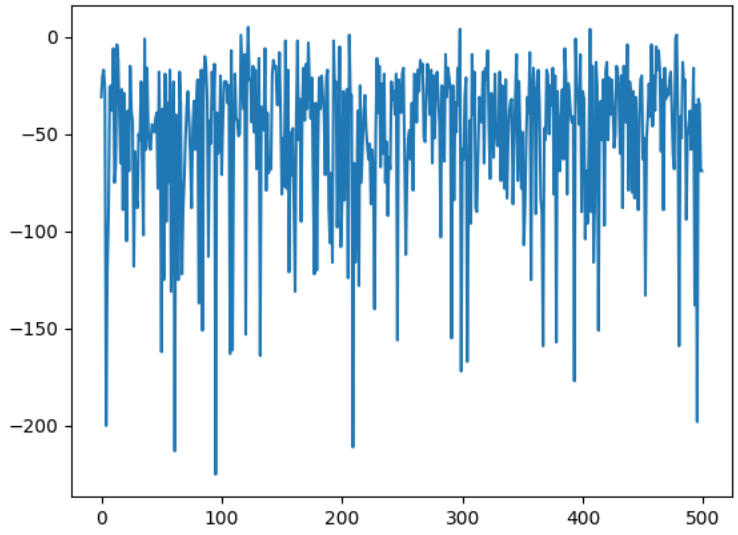


Figura 13- Resultados alcançados pelo Agente Aleatório no problema *"GridWorld"*

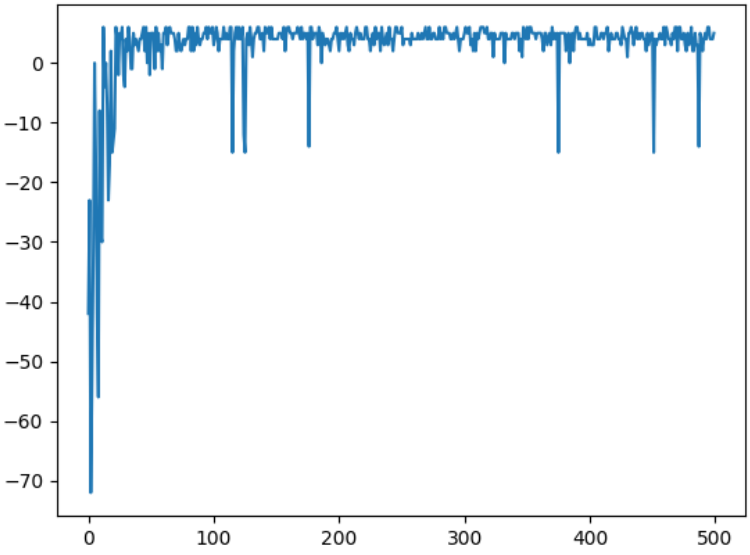


Figura 14- Resultados alcançados pelo Agente *Q-Learning* no problema *"GridWorld"*

2.9. Aplicação de ML em Meta-Heurísticas

Este trabalho tem como objetivo integrar técnicas de ML em Meta-Heurísticas para resolver Problemas de Otimização Combinatória, nos quais se incluem os Problemas de Escalonamento. Esta tarefa pode ser realizada de diferentes maneiras, o que torna essencial a análise de estudos existentes para compreender as diferentes técnicas envolvidas.

Nos últimos anos, o interesse pela investigação sobre o uso de algoritmos de ML em técnicas de otimização, como as Meta-Heurísticas, tem crescido significativamente. Mamaghan et al. [219] e Talbi [150] apresentam revisões exaustivas sobre o tema e propões diferentes categorias de classificação. Embora utilizem nomenclaturas distintas para cada categoria, o

conteúdo é similar e abordam os mesmos tópicos. A taxonomia de Mamaghan et al. [219] é composta por diversas categorias, incluindo Seleção do Algoritmo, Inicialização, Evolução, Avaliação do Desempenho, Definição de Parâmetros – Parametrização, e Cooperação. De seguida, algumas dessas categorias serão abordados de forma concisa.

Seleção do Algoritmo *(Algorithm Selection)*

No âmbito da aplicação de técnicas de otimização para resolver Problemas de Escalonamento, emerge frequentemente a questão sobre qual é o algoritmo que provavelmente apresentará o melhor desempenho, porque não existe um que seja capaz de apresentar sempre os melhores resultados independentemente do problema ou instância a resolver. Testar exaustivamente todos os algoritmos disponíveis é uma tarefa onerosa e, na prática, inviável devido às limitações impostas [219].

O desenvolvimento de uma abordagem que visa solucionar o Problema de Seleção de um Algoritmo requer informações detalhadas sobre o problema a ser resolvido, todos os algoritmos passíveis de seleção, bem como os critérios de desempenho, que são preponderantes para alcançar o objetivo: identificar o algoritmo que oferece o melhor equilíbrio entre as suas características e a capacidade de resolver o problema [220]. A Figura 15 apresenta um diagrama representativo de um Problema de Seleção do Algoritmo.

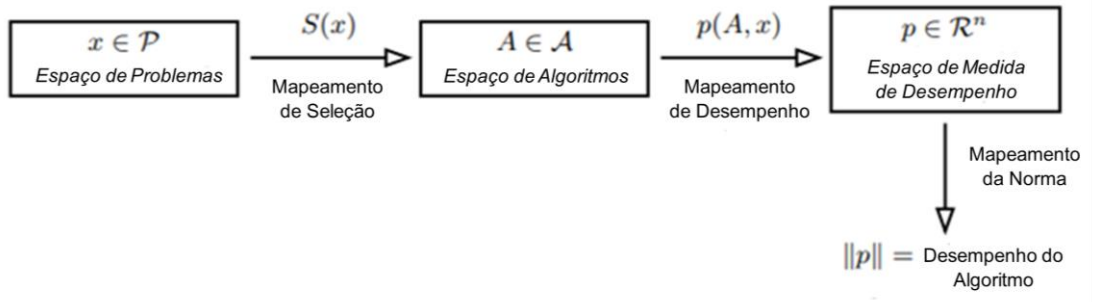


Figura 15- Representação de um Problema de Seleção do Algoritmo (adaptado de [220])

O espaço de algoritmos corresponde a um portefólio que contém os diferentes algoritmos. A sua construção pode ser classificada como estática ou dinâmica. Estático significa que o portefólio é constituído por um conjunto de algoritmos fixos que são incluídos antes da resolução de qualquer problema, e não muda durante a sua resolução. Por sua vez, quando o portefólio é dinâmico, a lista de algoritmos bem como a sua configuração podem mudar durante a resolução de uma instância de um problema [221, 222].

Nesta temática, a utilização de técnicas de ML surge como uma alternativa plausível que visa selecionar automaticamente o(s) algoritmo(s) mais apropriado(s) para uma determinada instância de um problema. O processo de seleção do algoritmo a utilizar pode ser *online* ou *offline*. No caso de ser *offline*, utiliza-se um conjunto de instância de treino com o objetivo de prever o mapeamento problema-algoritmo para novas instâncias do problema [150, 219]. Por sua vez, quando é *online*, a aprendizagem acontece durante a pesquisa, o que significa que todo o protótipo de seleção é construído e aplicado dinamicamente durante a resolução das instâncias [223].

Quando o objetivo é prever o desempenho de cada algoritmo para cada instância do problema, existem diferentes técnicas de ML com ampla aplicação. Pode ser realizado através de técnicas de Classificação, no qual o modelo prevê qual é o melhor algoritmo entre k possíveis [150]. Podem ser mencionadas técnicas de Aprendizagem Supervisionada como *Support Vector Machine, Decision Tree, Artificial Neural Network* (ANN) e *Nearest Neighbors*. Recorrendo à Regressão, o desempenho de cada técnica é previsto através de um modelo de regressão que, posteriormente, seleciona o que apresenta o melhor desempenho. Um dos métodos mais propostos é a *Linear Regression*, utilizada no estudo proposto em [224].

Quando o processo é *online*, os algoritmos de RL têm-se destacado. De forma geral, uma formulação de um problema é com as Meta-Heurísticas a representarem ações, os estados a corresponderem às soluções, o ambiente a ser representado pela instância do problema a resolver, e a recompensa a traduzir o desempenho da Meta-Heurística [225].

Inicialização *(Inicialization)*

Um passo fundamental no processo de implementação das Meta-Heurísticas é a geração de uma solução inicial ou de uma população de soluções, dependendo da técnica adotada. Esta etapa de inicialização das Meta-Heurísticas pode influenciar significativamente o desempenho da técnica, com alguns algoritmos a mostrarem uma maior sensibilidade ao processo de inicialização do que outros, conforme é evidenciado no estudo apresentado em [226]. Em determinados casos, pode ocorrer uma convergência prematura e a Meta-heurística ficar presa em ótimos locais caso não exista diversidade nas soluções iniciais, ou pode ser necessário executar um número maior de iterações se a inicialização ocorrer a partir de soluções com baixa qualidade, aumentando assim o tempo de execução do algoritmo [219, 227].

Uma abordagem clássica consiste na geração aleatória de soluções, especialmente quando existe ausência de informações prévias sobre o problema a resolver (ou, de forma mais específica, do espaço de soluções). Por exemplo, para um problema *Job-Shop* com n atividades e m máquinas, são gerados números inteiros, compreendidos entre 1 e n, e cada valor é repetido m vezes. A ordem por qual aparecem é aleatória. Em [228] podem ser consultados exemplos comuns para o problema TSP. Este processo é assim adaptado para incluir as características do tipo de problema em questão, mas a base e os princípios são os mesmos. Neste cenário, a qualidade da solução não é tida em consideração, o que pode resultar, em geral, em soluções de qualidade mais fraca [227, 229]. Em contrapartida, outras técnicas conseguem garantir a qualidade das soluções iniciais. Neste caso, a aplicação de ML surge com o intuito de produzir soluções iniciais de qualidade, mantendo uma diversidade adequada e, sempre que possível, reduzir os custos computacionais [150, 219].

O principal objetivo é extrair conhecimento da resolução de problemas anteriores para gerar soluções de boa qualidade. As técnicas de ML podem ser utilizadas para aprender propriedades das melhores soluções já utilizadas para resolver casos semelhantes. *Case-Based Reasoning* (CBR), Técnicas de *Clustering* e *Neural Networks* são algumas das técnicas com ampla aplicação que são incorporadas nas abordagens desenvolvidas neste domínio. De forma geral, inicialmente selecionam-se instâncias de treino e obtêm-se boas soluções para esse conjunto. As técnicas de ML extraem características das melhores soluções e utilizam-nas para gerar

soluções iniciais de qualidade para a próxima instância a ser resolvida, com base em algumas métricas definidas no processo. Em [150] pode ser visualizado um diagrama representativo desta metodologia de aprendizagem. Tendo em conta os princípios subjacentes a este tipo de abordagem, pretende-se, tal como Talbi [150] menciona, adquirir e utilizar conhecimento suficiente que permita intensificar a pesquisa num espaço de soluções próximo de uma solução promissora, conhecido como intensificação.

Numa vertente diferente, surge a utilização de técnicas de ML com foco na exploração de diferentes regiões do espaço de pesquisa. Neste contexto, a estratégia é dividir o espaço de pesquisa em vários subespaços, e, para cada um, gerar uma solução inicial. Em técnicas de ML como a *Multi-Armed Bandit*, cada braço é representativo de uma região do espaço de pesquisa e o objetivo é que a mesma seja capaz de selecionar as regiões que devem ser exploradas, ou seja, consiga selecionar as soluções iniciais de uma forma mais inteligente [230, 231].

Evolução do Processo de Pesquisa *(Evolution)*

Este domínio engloba todo o processo de pesquisa, desde a geração da solução inicial até ao momento onde a melhor solução é encontrada, e pode apresentar uma ampla aplicação dependendo do tipo de algoritmo alvo. As técnicas de ML são frequentemente utilizadas com o objetivo de orientar o processo de busca das técnicas de otimização, de forma a permitir que estas sejam capazes de melhorar o processo de geração de soluções vizinhas [219, 232].

Existem Meta-Heurísticas que utilizam operadores, como crossover, mutação e seleção, para explorar o espaço de soluções. A execução de um operador provoca, normalmente, uma alteração numa solução, no entanto, a eficácia de cada um pode variar ao longo da execução do algoritmo e em função do problema a resolver, o que pode afetar o bom desempenho do algoritmo em geral. Na literatura existem técnicas que visam selecionar dinamicamente os operadores mais apropriados para serem aplicados durante a execução do algoritmo [232]. O passo inicial de uma abordagem típica consiste na seleção do(s) operador(es) e na definição dos critérios de desempenho, que pode ser a medição da qualidade das soluções, a velocidade de convergência, entre outras. Após a execução, é calculada a recompensa, o que significa monitorizar o desempenho de cada operador. Normalmente é atribuído um crédito a cada operador, que é baseado nas recompensas, ou seja, na análise dos resultados em termos de alteração nos critérios de desempenho. De seguida, é então selecionado o operador a aplicar na iteração seguinte, de acordo com vários métodos disponíveis, como a Seleção Aleatória, a Seleção de Créditos Máximos, entre outros [233, 234].

O modo como uma decisão é tomada numa estratégia de Seleção de Operadores Adaptativa pode apresentar diferentes vertentes: com base no desempenho histórico dos operadores durante o processo de otimização, sendo comum, por exemplo, apresentarem a tendência para selecionar o operador com o melhor desempenho histórico; ou através das relações entre as características da pesquisa (como características das soluções atuais, sobre o processo de otimização em si, entre outros) e dos melhores operadores para o estado em análise [235, 236].

As técnicas de ML são integradas nestas abordagens para ajudar na seleção dos operadores, frequentemente através da utilização de informação (feedback) sobre o desempenho de cada

um. Neste caso, tendo em conta a natureza do feedback, a Aprendizagem por ser *Offline* ou *Online*. A Aprendizagem *Offline* caracteriza-se pela extração de informação a partir de um conjunto de instâncias de treino, que é utilizada para selecionar os operadores em novas instâncias do problema. Aydin et al. [237] analisaram a eficácia da Aprendizagem *Offline* na resolução de diferentes instâncias e vários tipos de problemas. Numa fase inicial, o algoritmo *Artificial Bee Colony* é implementado com um esquema de seleção de operadores e é executado para resolver uma determinada instância de um problema. Posteriormente, através de *Transfer Learning*, um subcampo do RL, um Agente é treinado para resolver instâncias do mesmo problema com dimensões diferentes, e instâncias de outros problemas típicos, ou seja, o objetivo é preservar a informação obtida através da resolução de um tipo de problemas para aplicar noutros que representam contextos diferentes. Apesar de apresentar resultados promissores, principalmente em instâncias de problemas similares, este tipo de aprendizagem pode ser limitado quando submetido à resolução de instâncias de problemas diferentes.

Num processo de Aprendizagem *Online*, o conhecimento é recolhido e utilizado dinamicamente durante a resolução [223, 238]. Quando o RL ou o *Deep Reinforcement Learning* (DRL) são integrados, o processo de seleção iterativa do operador é formulado como um Processo de Decisão de Markov, onde cada operador corresponde a uma ação que pode ser escolhida em cada estado, e a função de recompensa avalia o desempenho do operador [233]. Em [239], os autores propõem um método de seleção de operadores baseado em RL para Problemas de Otimização Multiobjectivo. Um Agente utiliza *Deep Neural Networks* para aprender uma política que lhe permita estimar o valor de Q de cada ação num determinado estado, valor este que representa melhoria cumulativa esperada do desempenho de um operador. O objetivo foi assim aprender as relações entre as variáveis de decisão e os valores Q dos operadores candidatos durante o processo de otimização, o que foi um sucesso pois a abordagem demonstrou ser capaz de selecionar o melhor operador para cada problema. Yi et al. [240] utilizam dois métodos baseados em RL, o *Deep Q-Learning* e *Proximal Policy Optimization* (PPO), para desenvolver uma estrutura que é capaz de selecionar e combinar, de forma inteligente, os operadores em diferentes fases do processo de otimização. Em [241], é o *Double Deep Q-Learning* que é utilizado com sucesso para controlar as estratégias de mutação de Algoritmos Evolucionários, como *Differencial Evolution*. Numa vertente diferente e inovadora, Pei et al. [233] combinaram os dois tipos de aprendizagem para desenvolver um *Framework*, constituído por dois módulos de seleção de operadores, e com uma política de decisão adaptativa focada na manutenção do equilíbrio entre os dois. Outros trabalhos que merecem ser destacados podem ser consultados em [235, 242, 243]. Em [219, 233] podem ser consultadas estudos que apresentam alguns dos trabalhos publicados com sucesso sobre a temática da seleção dos operadores.

Avaliação do Desempenho *(Fitness Evaluation)*

A avaliação do desempenho (ou aptidão) refere-se ao processo de medir a qualidade ou adequação de uma solução candidata ao problema em resolução. Neste caso é atribuído um valor de qualidade a cada solução, que é usado para comparar e classificar as soluções, guiando o processo de procura pelo melhor resultado possível [219].

Em diversos Problemas de Otimização não existe uma função de desempenho analítica através da qual as soluções são avaliadas, ou, quando existe, pode ser necessário um grande número de avaliações, o que se torna difícil e computacionalmente dispendioso [244 – 246]. Um exemplo do aparecimento deste tipo de problemas é durante a aplicação de GA, onde a possibilidade de existir um elevado número de indivíduos numa população pode ter como consequência a necessidade de um custo demasiado elevado com a avaliação do desempenho de cada um, o que se agrava à medida que a dimensão da população aumenta [247].

Como alternativas para solucionar este tipo de problemas, existe uma variedade extensa de métodos que são aplicados. Dependendo do tipo de algoritmos ou técnicas com que são aplicados ou integrados, o modo como são agrupados ou classificados pode ser um elemento diferenciador entre os trabalhos apresentados pelos diversos autores. Uma estratégia comum na literatura é a utilização de aproximações computacionalmente eficientes da função de avaliação de desempenho, num processo conhecido como *Fitness Approximation* [246, 248]. Neste processo, os valores de desempenho dos indivíduos são estimados sem efetuar a avaliação computacionalmente dispendiosa do desempenho para cada um. A função de desempenho ou aptidão é substituída por um modelo aproximado que imita o comportamento da função original. De forma geral, estas técnicas são treinadas utilizando um conjunto de dados de treino que incluem soluções candidatas e os seus valores de desempenho associados, isto é, em que a variável de entrada é um conjunto de características extraídas das soluções e a variável de saída é o valor de desempenho original de cada solução [219, 246, 249].

Os *Surrogate Models* – Modelos Substitutos, são um exemplo particular, onde o modelo aproximado (substituto) é refinado iterativamente durante o processo de pesquisa [219]. Estes modelos simulam o comportamento da função de desempenho real, mas avaliam muito mais rapidamente, reduzindo assim o custo computacional associado ao método de otimização [250, 251]. Jin [246] menciona a utilização destes modelos num processo de aprendizagem *Offline* e *Online*, que pode englobar técnicas como *Linear Regression*, ANNs e SVMs. As ANNs capturam relações complexadas e não lineares entre variáveis de entrada e valores de desempenho. Horng et al. [252] utilizam ANN como *Surrogate Models* para a aproximação de funções de desempenho em Problemas de Otimização Estocástica por Simulação. A ANN é empregue como Modelo Global Substituto. Inicialmente, o modelo é treinado usando um subconjunto representativo do espaço de entrada, cujos dados são normalizados para reduzir o erro de estimativa e o tempo de treinamento. O processo envolve o ajuste dos pesos da rede neural para minimizar o erro entre as previsões da ANN e os valores exatos de desempenho. Após a finalização do treino, durante a fase global de busca, um EA é usado para explorar o espaço de soluções. O EA gera uma população de soluções, e em vez de se avaliar cada uma diretamente usando um modelo exato, o conjunto do Modelo Global Substituto é utilizado para fornecer uma aproximação do desempenho de cada solução. A ideia central deste trabalho é assim treinar as ANNs para fornecer estas estimativas rápidas, reduzindo o tempo computacional necessário de forma significativa.

Um modelo SVM apresenta a capacidade de modelar a função de desempenho em espaços de alta dimensão, o que significa que se destaca quando existe uma clara separação entre classes. Quando comparados com outros modelos de aproximação, este não é tão sensível a ótimos

locais e o seu processo de otimização não depende das dimensões do problema [249]. Rosales-Pérez et al. [253] propõe uma abordagem que combina um EA com um conjunto de *Surrogate Models* baseados em SVMs, que são usados para aproximar as funções de desempenho de Problemas de Otimização MultiObjectivo. Os SVMs são treinados utilizando amostras de treino derivadas das soluções armazenadas num arquivo externo, com a seleção de hiperparâmetros a ser realizada através de *Grid Search* e Validação Cruzada *K-Fold*. Estes modelos são atualizados iterativamente para incorporar novos dados e são utilizados para avaliar inicialmente os descendentes, reduzindo o número de avaliações das funções de desempenho reais e melhorando a eficiência computacional, enquanto mantêm a precisão das estimativas de desempenho.

Numa vertente direcionada para lidar com um EA, surge o conceito de *Evolutionary Approximation*, no qual o objetivo com a aproximação evolutiva é reduzir o esforço computacional através da aproximação dos elementos dos EAs [246]. Existem duas subcategorias de eleição: *Fitness Inheritance*, no qual as avaliações do desempenho de um individuo são calculadas a partir do valor do desempenho dos seus progenitores [254, 255], e *Fitness Imitation*, no qual os indivíduos são agrupados em grupos, apenas um indivíduo por grupo é avaliado através da respetiva função, e o valor dos outros indivíduos do mesmo grupo é estimado com base no valor do indivíduo representativo do seu grupo [246, 256]. Em [249], Shi e Rasheed apresentam uma revisão extensa de métodos adotados com especial foco no contexto dos EAs, e apresenta exemplos de aplicação para as diferentes categorias que os autores usam para classificar as diferentes técnicas.

Em suma, após a análise das diversas formas de integração, é evidente que são utilizadas com o objetivo de reforçar a qualidade das Meta-Heurísticas, torná-las em técnicas mais eficientes, e melhorar o seu desempenho em aspetos como robustez, capacidade de exploração e qualidade da solução, entre outros.

Devido à complexidade do tema e à existência de várias formas de integração, é uma tarefa difícil definir uma *query* única e abrangente, capaz de apresentar todos os artigos relevantes para cada categoria de classificação. Sendo assim, dado o foco deste trabalho, a revisão bibliográfica será concentrada na análise da integração de técnicas de ML para parametrizar as Meta-Heurísticas, ou seja, na categoria *Parameter Setting*, o que faz com que esta forma de integração apenas seja abordada na fase seguinte. No Apêndice B pode ser consultada um trabalho desenvolvido para a Unidade Curricular de Metodologias de Investigação e Planeamento, que explora a aplicação de algoritmos de ML na resolução de Problemas de Escalonamento. O estudo inclui uma análise bibliométrica, uma revisão de estado da arte através do estudo de casos relevantes, e uma análise crítica focada na identificação de ideias que podem ser aplicadas em pesquisas futuras sobre o tema.

2.10. Análise Bibliométrica

Para perceber a relevância e o potencial impacto do trabalho proposto nesta dissertação, é fundamental entender as contribuições passadas e presentes que se encontram na literatura, e identificar padrões, lacunas e tendências emergentes. Assim, não só se posiciona o trabalho

no contexto atual, como também se constrói um conhecimento aprofundado sobre potenciais aspetos que o mesmo pode explorar [257].

Uma revisão da literatura é um processo metódico que tem o objetivo de analisar o estado de arte de um determinado domínio temático [258]. O principal objetivo da análise neste trabalho é perceber como é que os algoritmos de ML têm sido utilizados para parametrizar as Meta-Heurísticas, ou seja, quais são as estratégias adotadas para definir os valores dos parâmetros de cada uma. Neste caso, para a recolha de dados, isto é, obter o conjunto de publicações a analisar, utilizou-se a base de dados *Web of Science* porque é reconhecida como um dos maiores repositórios de documentos científicos, o que significa que contém uma vasta coleção de publicações de investigação.

A *query* foi definida especificamente para a categoria de classificação de Parametrização - *Parameter Setting*, e é constituída por três conjuntos de palavras-chave, sendo importante mencionar o seguinte:

- → O primeiro conjunto relaciona-se com a forma de integração. Para o caso da Parametrização, utilizam-se sete dos termos mais comuns usados na literatura – *parameter setting, parameter calibration, parameterization, parameter tuning, parameter control, offline tuning* e *online tuning*, de forma a obter uma pesquisa inclusiva. O objetivo é incluir os diferentes modos de referir o processo em si, mas também os tipos de parametrização – *Offline* e *Online*;
- → O segundo conjunto de palavras-chave é dedicado às técnicas de otimização, que são as Meta-Heurísticas. Este termo pode ser representado de diferentes formas, e, após um período de experiências e análises, verificou-se que referir a palavra no singular ou no plural alterava significativamente a quantidade de publicações obtidas, logo são incluídas todas as possibilidades;
- → O terceiro conjunto relaciona-se com as técnicas de ML utilizadas. Neste caso são incluídos os diferentes tipos de aprendizagem – *Supervised Learning, Unsupervised Learning* e *Reinforcement Learning*, para incluir as publicações que usem termos mais específicos que a palavra mais comum – *Machine Learning*. O mesmo se aplica a *"Data Mining"* e *"Prediction Methods"*, que foram introduzidas após as suas presenças serem detetadas, de forma significativa, na análise de várias publicações sobre este tema;
- → Para todas as Palavras-Chave aplica-se o termo TOPIC, o que conduz a pesquisa para os campos mais importantes, neste caso título, resumo e palavras-chave.

A Tabela 7 apresenta assim *query* usada, onde os termos estão em inglês, e os resultados obtidos, em termos de número de publicações relacionadas com o tema deste trabalho, da pesquisa realizada durante o mês de julho de 2024. Tendo em conta o detalhe que se pretende obter, com a análise exclusiva da aplicação de técnicas de ML para parametrizar Meta-Heurísticas, o número de publicações é 368.

Tabela 7- Pesquisa por Palavras-Chave

Base de Dados	*Query*	Resultados
Web of Science	TOPIC ((“Parameter Setting” OR “Parameter Calibration” OR “Parameterization” OR “Parameter Tuning” OR “Parameter Control” OR “Offline Tuning” OR “Online Tuning”) AND (“Metaheuristic” OR “Metaheuristics” OR “Meta-Heuristic” OR “Meta-Heuristics” or “Simulated Annealing” or “Swarm Intelligence” or “Genetic Algorithm” or “Genetic Algorithms” OR “Evolutionary Algorithms”) AND (“Machine Learning” OR “Artificial Intelligence” OR “Supervised Learning” OR “Unsupervised Learning” OR “Reinforcement Learning” OR “Deep Learning” or “Data Mining” or “Prediction Methods”))	368

Para analisar e avaliar quantitativamente as publicações utilizaram-se as seguintes dimensões: evolução temporal das publicações e citações, áreas de estudo com mais publicações e coocorrência das palavras-chave utilizadas. O tratamento e mapeamento de dados foi realizado recorrendo ao software VOSViewer, nomeadamente para avaliar esta última dimensão [259].

A Figura 16 revela a evolução do interesse no tema desta dissertação entre 19989 e julho de 2024, sendo que a opção por incluir todos os anos possíveis com publicações tem como objetivo perceber o momento no qual este tema se tornou alvo de estudo pela primeira vez. Os primeiros anos analisados assumem-se como os menos produtivos, com apenas 1 publicação anual. Apesar de existir um ligeiro aumento, os números de publicações permanecem relativamente baixos até ano de 2018, no entanto, a partir de 2019 evidencia-se um aumento notável, com o número de publicações a superar sempre o ano anterior, indicando assim uma crescente relevância deste tema. Os anos de 2022 e 2023 posicionam-se nos dois primeiros lugares, com 2023 a ser o mais produtivo com 74 publicações.

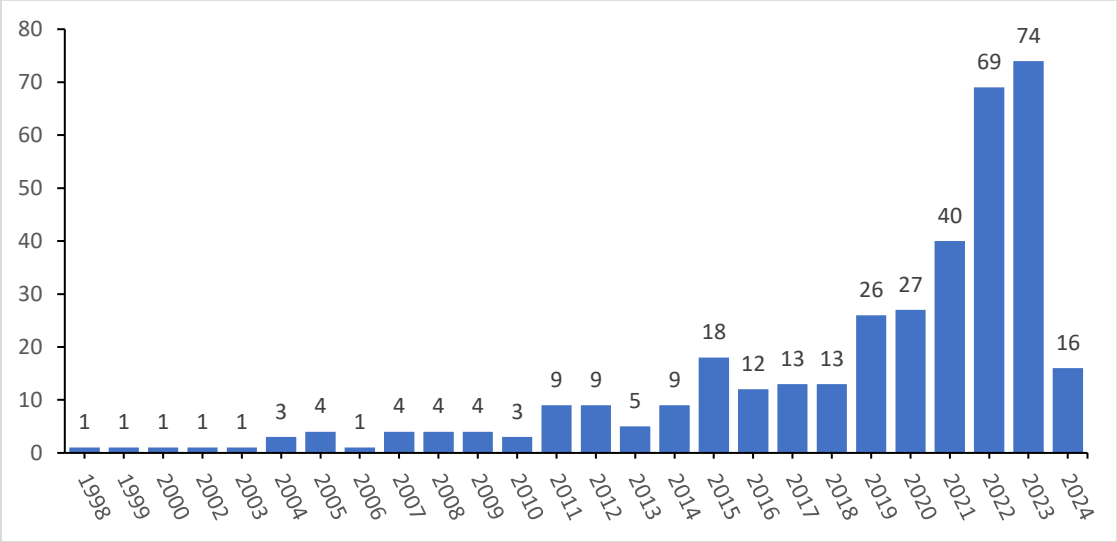


Figura 16- Distribuição temporal das publicações entre 1998 e julho de 2024

A análise das áreas de investigação a que as publicações se associam é importante para caracterizar o campo de investigação. Os resultados apresentados na Figura 17 mostram que,

das categorias do *Web of Science*, a "Ciência da Computação, Inteligência Artificial" é a mais frequente, o que se enquadra no tema pois compreende áreas que fornecem uma base teórica e prática para o desenvolvimento de algoritmos, tanto de ML como de técnicas de otimização. A "Engenharia Elétrica e Eletrónica", a "Ciências da Computação, Métodos Teóricos" e "Ciências da Computação, Sistemas de Informação" posicionam-se de seguida, não só complementa a categoria com maior destaque como também sugerem áreas específicas, e, no caso da última mencionada, esta sublinha a importância da otimização no tema deste trabalho. A diversidade de áreas de investigação evidencia assim a natureza multidisciplinar do tema.

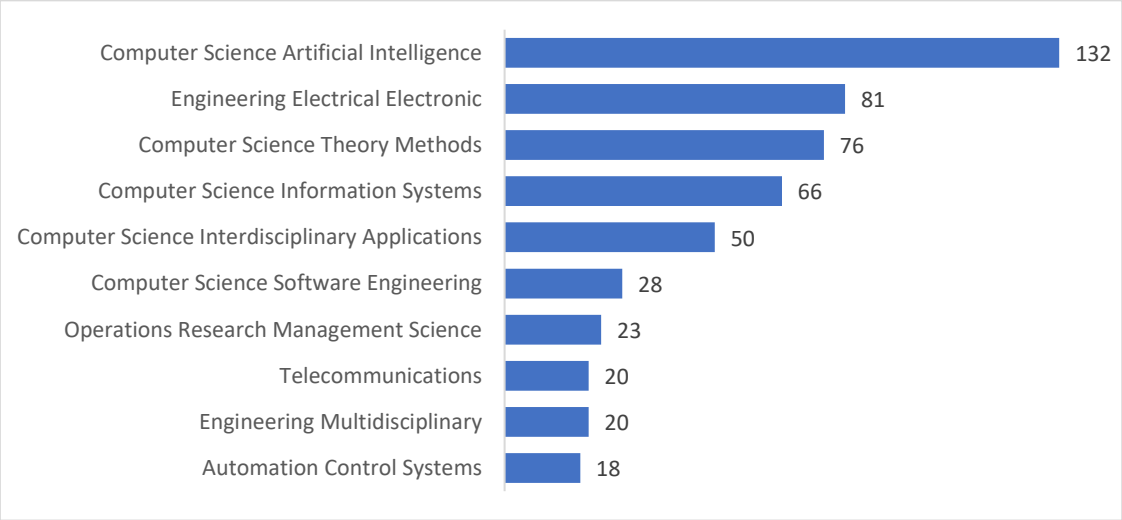


Figura 17- As dez principais áreas de investigação das publicações

A Figura 18 apresenta o mapa representativo da coocorrência e relevância das palavras-chave de cada publicação, que apresenta cinco clusters. *"Optimization"* é a palavra que apresenta a maior ligação com as restantes palavras *(total link strength)*, seguida de *"Machine Learning"* e *"Genetic Algorithm"*. Esta última palavra mencionada destaca-se por ser a que apresenta o maior número de ocorrências. De seguida surge *"Classification", "Algorithm", "Metaheuristics"* e *"Particle Swarm Optimization"*, todas praticamente no mesmo nível. Estes são termos com grande relevância pois a aplicação de algoritmos/técnicas de ML para parametrizar as Meta-Heurísticas é o cerne do trabalho. No caso do PSO e do GA, que se encontram num patamar acima, os seus aparecimentos posicionam estes algoritmos como sendo daqueles que apresentam maior aplicação. Por fim, numa perspetiva de análise às tendências e oportunidades futuras, expressões como *"Deep Learning"* e *"Reinforcement Learning"* são relevantes porque representam abordagens que se tornaram tendências recentes de investigação por parte da comunidade científica. De maneira geral, o mapa mostra que todos os conceitos estão direta ou indiretamente relacionados.

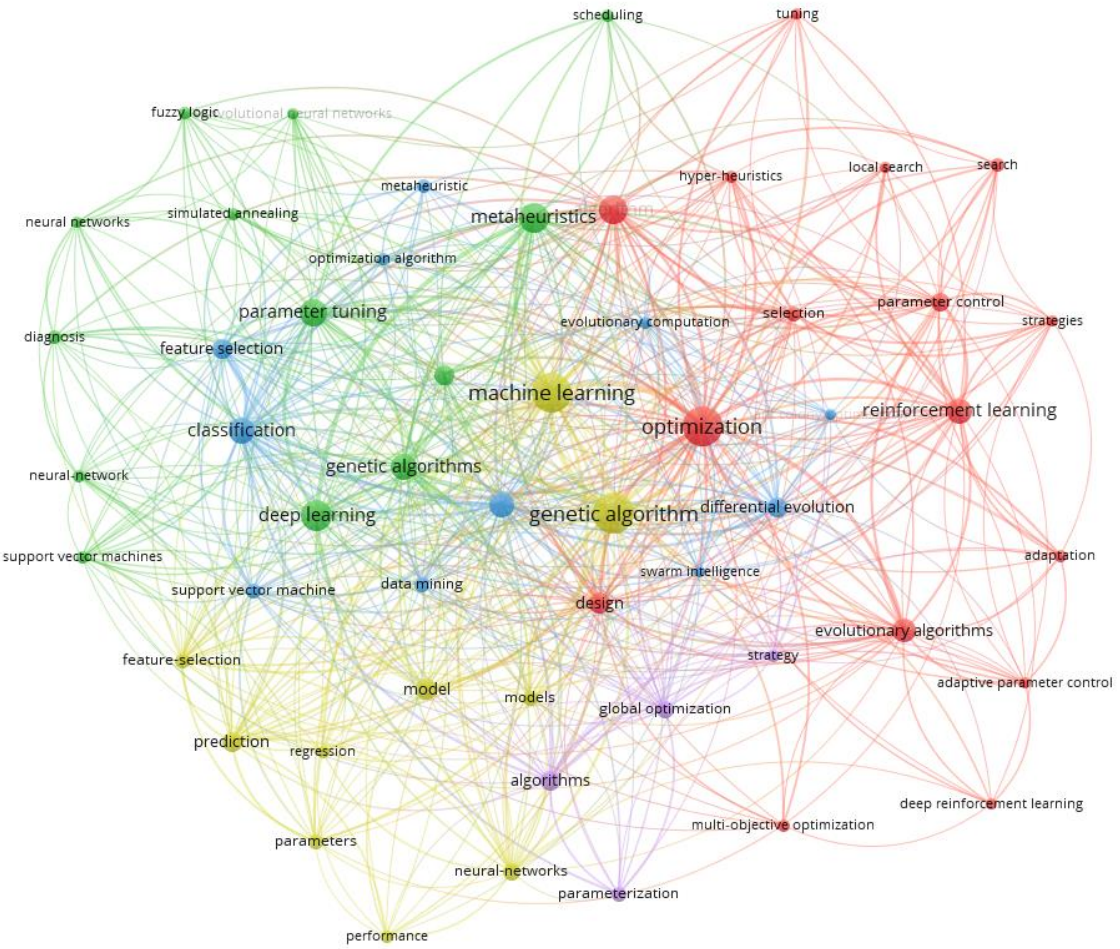


Figura 18- Mapa de Coocorrência de Palavras-chave

2.10.1. Revisão do Estado da Arte - Estudo de Casos

O facto de o ML capacitar sistemas a aprender e melhorar tem transformado a maneira como abordamos e resolvemos problemas complexos. No contexto do Escalonamento, observa-se uma crescente aplicação dos diferentes tipos de Aprendizagem para melhorar o desempenho das técnicas de otimização, como as Meta-Heurísticas.

Para os vários artigos focados na aplicação de técnicas de ML para parametrizar Meta-Heurísticas, é possível identificar diferentes características específicas que os caracterizam, como o tipo de Parametrização, a técnica de ML utilizada, a Meta-Heurística que é o alvo de estudo e os respetivos parâmetros. A Tabela 8 apresenta a informação mencionada para 18 artigos identificados que se destacam pela sua excelente qualidade em termos de relevância para o tema desta dissertação.

Tabela 8- Classificação de 18 artigos relevantes identificados

Autor	Tipo de Parametrização	Técnica de ML	Meta-Heurística	Parâmetro	Tipo de Problema (Dimensão)
Tatsis e Parsopoulos [260]	*Online*	Algoritmo REINFORCE	*Differential Evolution* (DE)	Parâmetros do *Gradient-Based Parameter Adaptation with Line Search:* $t_{sec}, t_{pri}, \lambda$	Problemas de Otimização em Grande Escala (entre 50 e 200)
Lessmann et al. [261]	*Online*	*Regression Models* (SVM e *Linear Regression*)	*Particle Swarm Optimization* (PSO)	*Learning rates* (c_1, c_2) e *maximum velocities* (velocidades máximas) - $v_{max_{factor}}$	Problema do domínio do Planeamento da Rede de Abastecimento de Água
Huynh et al. [262]	*Online*	RL: *Q-Learning*	*Differential Evolution (DE)*	*Scale Factor* (F) e *Crossover Rate* (C_r)	Problemas de Otimização Estrutural com Restrições (Treliças)
Chaves e Lorena [263]	*Online*	RL: *Q-Learning*	*Biased Random-Key Genetic Algorithm (BRKGA)*	Q-Learning: *Learning Rate* (α), *Discount Factor* (γ), e *Epsilon* (ε) BRKGA: *Population Size, Percentage of Elites* e *Percentage of Mutants*	*Traveling Salesman Problem* (TSP)
Pikalov e Pismerov [264]	*Online*	*Feedforward Neural Network*	$(1 + (\lambda, \lambda))$ *Genetic Algorithm*	*Mutant phase population size* (λ_1), *crossover phase population size* (λ_2), *mutation coefficient* (k), *crossover probability* (c)	*W-Model Problem* (n= 256 e n=512)
Smirnov e Mironovich [265]	*Online*	*Feedforward Neural Network*	$(1 + (\lambda, \lambda))$ *Genetic Algorithm*	*Mutant Phase Population Size* (λ_1), *Crossover Phase Population Size* (λ_2), *Mutation Coefficient* (k), *Crossover Probability* (c)	*W-Model Problem* (n= 24 e n=32)
Reijnen et al. [266]	*Online*	RL: *Proximal Policy Optimization* (PPO)	*Genetic Algorithm* (GA)	*Mutation Probability* e *Crossover Probability*	*Multi-Objective Job-Shop Scheduling Problems*
Caselli et al. [267]	*Online*	*Density-Based Spatial Clustering of Applications with Noise*	*Cuckoo Search Algorithm*	*Number of Nests* (Número de Ninchos) e *Probability Abandon Nest* (Probabilidade de Abandono dos Ninchos)	*Set Covering Problem*
Aleti et al. [268]	*Online*	*Linear Regression*	*Evolutionary Algorithms* (EAs)	Parâmetros dos EAs: *mutation rate, crossover rate, population size, matting pool size, mutation operator, crossover operator*	*Quadratic Assignment Problem*
Sun et al. [269]	*Online*	RL: *Policy Gradient* (PG)	*Differential Evolution* (DE)	*Scale Factor* (F) e *Crossover Rate* (C_r)	Conjuntos de testes CE'13 e CE'17
Karafotias et al. [270]	*Online*	RL*: Temporal Difference Learning, Dynamic State Space Segmentation* e *Eligibility Traces*	Quatro tipos de EA: *Simple ES, Cellular GA, GA with Multi-Parent Crossover* (GA MPC) e *IPOP-10DDr CMA ES*	Parâmetros característicos de cada tipo de EA	Simples ES: Rastrigin, Schaffer e Fletcher & Powell; Cellular GA e IPOP-10DDr CMA ES: BBOB $f_{21}, f_{22}, f_{23}, f_{24}$; GA MPC: CEC2011 $f_7, f_{11.8}, f_{12}, f_{13}$
Lacerda et al. [271]	*Online*	RL: *Twin Delayed Deep Deterministic Policy Gradient*	*Heterogeneous Comprehensive Learning Particle Swarm Optimization* (HCLPSO), *Fish School Search* (FSS), DE, *Binary GA and Ant Colony Optimization* (ACO)	Parâmetros característicos de cada MH utilizada	HCLPSO, FSS e DE: CEC17 *Continuous Functions*; *Binary GA: Knspsack Problem*; ACO: TSP

Gomez e Toosi [272]	*Online*	RL*: Deep Deterministic Policy Gradient* (DDPG) e *Proximal Policy Optimization* (PPO)	*Genetic Algorithm* (GA)	*Mutation Probability* e *Crossover Probability*	*N-Queen* (n= 8, 12 e 15), *OneMax* (n= 30, 35 e 40) e *Password Cracker* (n= 8, 10 e 12)
Bora et al. [273]	*Online*	*Reinforcement Learning*	*Non-Dominated Sorting Genetic Algorithm*	*Crossover and Mutation Probabilities and Indices*	*Satellite Coverage Problem (Optimization of Broad-Band Reflector Antennas Satellite)*
Pavón et al. [274]	*Offline*	*Bayesian Networks* (com *Case-Based Reasoning* - CBR*)*	*Genetic Algorithm* (GA)	*Mutation Probability, Crossover Probability* e *Population Size*	*Constraint-based Geometric Design Problem*
Šilc et al. [275]	*Offline*	*Predictive Clustering Trees*	*Differential Ant-Stigmergy Algorithm* (DASA)	Parâmetros do DASA: m, p, b, s_+, s_-	-
Pereira e Madureira [276]	*Offline*	*Racing Based Learning Model (F-Race Method)*	TS, GA, SA, ACO, PSO, ABC	Parâmetros do TS, GA, SA, ACO, PSO, ABC	*Extended Job-Shop Scheduling Problem*
Tondut et al. [277]	*Offline*	*Kriging Methodology*	*Particle Swarm Optimization, Differential Evolution* (DE) *e Clonal Selection Algorithm* (CLONALG)	PSO: *Number of Particles* e *Learning Rates*; DE: *Population Size*, *Scale Factor*, *Crossover Rate* e *Strategy*; CLONALG: Population Size of Antibodies, *Number of Antibodies Selected for Cloning*, *Multiplication Factor* e *Mutation Rate*	12 Funções Matemáticas (5 unimodais e 7 multimodais)

Da análise da Tabela 8, a revisão dos trabalhos focados na aplicação de técnicas de ML para a parametrização de Meta-Heurísticas revelou tendências e avanços significativos na área. A maior parte dos estudos concentra-se na Parametrização *Online*, onde os parâmetros são ajustados dinamicamente durante a execução do algoritmo, em vez de serem definidos previamente. Dentro deste espectro, o uso de RL destaca-se consideravelmente. Este tipo de aprendizagem permite que o algoritmo aprenda a ajustar os parâmetros da Meta-Heurística com base no *feedbback* recebido durante a interação com o ambiente, levando a uma melhoria contínua de desempenho. Desta forma, através de algoritmos como *Q-Learning* e PPO, é possível dotar um Agente para aprender e tomar decisões sequenciais em ambientes dinâmicos, características que são essenciais para a parametrização eficiente de Meta-Heurísticas. Além disto, outras técnicas como *Bayesian Networks, Predictive Clustering Trees* e *Linear Regression*, também têm sido exploradas, trazendo cada uma as suas próprias vantagens e contribuindo para a evolução do campo em estudo de maneira única.

Para uma compreensão detalhada do funcionamento de uma abordagem que utiliza técnicas de ML para parametrizar Meta-Heurísticas na resolução de Problemas de Otimização Combinatória, serão apresentados, de seguida, vários casos de estudos focados no tema.

Parametrização *Online: Reinforcement Learning* e *Deep Reinforcement Learning* para Parametrizar as Meta-Heurísticas

Chaves e Lorena [263] apresentam uma estrutura que combina o *Biased Random-Key Genetic Algorithm* (BRKGA) com o algoritmo *Q-Learning* (BRKGA-QL). BRKGA é uma Meta-Heurística utilizada para resolver Problemas de Otimização Combinatória, que é uma variação do algoritmo *Random Key Genetic Algorithm*. À semelhança de qualquer Algoritmo Genético, o objetivo nesta Meta-Heurística é melhorar uma população de cromossomas (ou soluções) em cada iteração através da aplicação de operadores genéticos.

O processo de evolução do BRKGA-QL é igual ao processo clássico que caracteriza o BRKGA. A particularidade inerente ao mesmo é o modo como o *Q-Learning* é integrado. Neste caso, este algoritmo controla, durante o processo evolutivo, todos os parâmetros envolvidos no BRKGA-QL, o que engloba os parâmetros do próprio algoritmo - fator de aprendizagem, fator de desconto e *epsilon*, e três parâmetros do BRKGA - *population size, percentage of elites* e *percentage of mutants*. É considerado um estado único para cada parâmetro, enquanto as ações seguem valores recomendados da literatura, exceto para o *population size*, que é definido com base na Sequência de Fibonacci. Em cada momento, se a configuração de parâmetros selecionada melhorar a solução, a recompensa é 1 e procede-se à atualização dos valores Q de acordo com os princípios que caracterizam a implementação do *Q-Learning* [263].

Na avaliação da eficiência e da qualidade do BRKGA-QL, o método proposto foi testado na resolução de 50 instâncias do problema TSP, com as dimensões das mesmas a variarem entre 51 e 3795 cidades. Com o Critério de Interrupção sendo definido como o tempo de execução, que, no TSP, pode ser descrito como o número de cidades na instância, e cada instância sendo executada 20 vezes, os resultados computacionais evidenciaram que o sistema foi capaz de obter bons resultados, encontrando a melhor solução em 72% das instâncias. Para instâncias até 200 cidades, a resolução é bem-sucedida em poucos segundos, no entanto, para instâncias superiores, o sistema enfrenta dificuldades. O desvio percentual relativo médio foi de 0,4%, e o pior valor apresenta um desvio inferior a 1%, o que é significativamente satisfatório [263]. A aplicação do *Q-Learning* foi assim capaz de provocar uma melhoria significativa no BRKGA e fornecer a melhor configuração de parâmetros para uma determinada instância, com a tabela de valores Q a ser significativamente eficiente à medida que o método proposto é executado devido à capacidade de extrair conhecimento do processo em si.

Numa vertente semelhante, Reijen et al. [266] apresentam o DEMOCA (*Deep Reinforcement Learning Multi-Objective Control Algorithm*), que é um sistema responsável por controlar, de forma adaptativa, os valores dos parâmetros de um EA para resolver Problemas Multiobjetivo através da utilização de DRL. O problema em si é modelado através de um Processo de Decisão de Markov, com métricas normalizadas para medir a qualidade de um conjunto de soluções durante o processo de pesquisa. Os parâmetros controlados são a *mutation probability* e *crossover probability*, o que significa que as ações (contínuas) correspondem a possíveis alterações nos parâmetros, respeitando um intervalo definido de acordo com as recomendações reportadas na literatura. Para guiar o ajuste de cada parâmetro, a função de recompensa é baseada no Hiper-Volume, que mede a qualidade e a diversidade do conjunto de soluções. O objetivo é maximizar o Hiper-Volume ao longo do tempo.

O objetivo é que o Agente seja capaz de aprender uma política que lhe permita selecionar a melhor configuração de parâmetros com base na sua interação durante a fase de pesquisa. Para isso, o Agente é treinado recorrendo ao algoritmo PPO, que é uma técnica bastante popular, e envolve a execução de 100 000 passos. Para avaliar o DEMOCA, foram utilizadas 41 instâncias do *Flexible Job-Shop Scheduling Problem*, no qual o sistema apresentou uma eficiência computacional significativamente superior ao *Grid Search Multi-Objective Genetic Algorithm*, requerendo cinco vezes menos esforço computacional. O Agente alcançou assim um desempenho de excelência, sendo capaz de ajustar os parâmetros em tempo real, adaptando-se às mudanças na dinâmica do problema [266].

Parametrização *Online*: Outros tipos de Aprendizagem para Parametrizar as Meta-Heurísticas

Lessmann et al. [261] propõe a utilização de modelos de Regressão para determinar os parâmetros de uma Meta-Heurística, *Particle Swarm Optimization* (PSO), com base em dados históricos de execuções anteriores do algoritmo. A coleta de dados é realizada a partir das execuções iniciais do PSO, onde se registam as assinaturas/características das partículas (soluções candidatas) e o desempenho correspondente. As assinaturas das partículas incluem características como posições e velocidades atuais, histórico de desempenho, entre outros fatores que podem influenciar a escolha dos valores dos parâmetros ótimos. Enquanto estas assinaturas constituem as variáveis dependentes, os valores dos parâmetros do PSO - $c_1, c_2, V_{máx}$, são as variáveis independentes. Os dados recolhidos são pré-processados para serem utilizados nos modelos de regressão, o que pode incluir normalização, tratamento de valores ausentes, seleção de variáveis relevantes, entre outros. Para lidar com a alta dimensionalidade dos dados, são utilizadas técnicas de redução de dimensionalidade e seleção de características, o que permitem assim melhorar a eficiência e a precisão dos modelos.

Para cada parâmetro do PSO, são construídos modelos de regressão. Os autores testaram diferentes tipos de modelo, incluindo modelos lineares e não lineares: *Classical Multiple Linear Regression, Stepwise Linear Regression Model, NAIVE, Least-Square Support Vector Machine (linear e radial), Random Regression Forests* (REGFOR). Estes modelos são treinados utilizando os dados pré-processados, sendo cada modelo ajustado para prever um parâmetro específico do PSO. A avaliação do modelo é feita através da utilização de técnicas de validação cruzada para assegurar que os modelos não estejam sobreajustados aos dados de treino. O modelo escolhido é o que apresenta o melhor desempenho, ou seja, o que tem a melhor capacidade de capturar relações complexas entre as variáveis de entrada e os valores reais dos parâmetros. Neste caso, entre os diferentes modelos testados, os modelos não lineares, como o REGFOR, demonstrarem ser os mais adequados e com melhor desempenho em comparação com modelos lineares mais simples [261].

Após o processo de treino e validação, estes modelos podem ser incorporados como agentes de previsão para substituir a abordagem padrão de definição de parâmetros. Conforme o PSO executa as suas iterações, novos dados são continuamente coletados, o que inclui o desempenho das partículas já com os novos parâmetros previstos pelos modelos. Com base nessa informação, os modelos de regressão ajustam os parâmetros em tempo real num processo de Aprendizagem *Online*, já que o modelo é refinado continuamente para melhorar a

precisão das previsões, adaptando-se assim dinamicamente às mudanças nas condições do problema e no desempenho das partículas. Todo este processo incluiu uma avaliação da eficácia dos parâmetros previstos, no qual, se estes não estiverem a melhorar o desempenho do algoritmo, são realizados ajustes nos modelos de regressão, o que pode incluir alterações ou ajustes dos modelos de regressão, reavaliação das variáveis utilizadas, entre outros [261]. O sistema proposto neste artigo assume-se assim como uma solução eficiente que recolhe informação do algoritmo durante o processo de pesquisa e a utiliza para atualizar os valores dos parâmetros num mecanismo de auto adaptação. Isto permite incorporar informação valiosa que pode ser ignorada quando os valores dos parâmetros do algoritmo são definidos com valores fixos e não são alterados ao longo da sua execução.

Parametrização *Offline*

Em [274], os autores desenvolvem um sistema que combina *Bayesian Networks* com a metodologia *Case-Based Reasoning* para ajustar automaticamente os parâmetros de Meta-Heurísticas na resolução de diferentes instâncias de problemas. Este sistema baseia-se num modelo de ajuste básico concebido pelos mesmos autores no passado, no qual, através da análise dos dados disponíveis de execuções anteriores de um algoritmo, uma *Bayesian Network* (BN) capta as relações entre os parâmetros e o desempenho do algoritmo. Posteriormente, um processo de inferência na BN recomenda uma configuração de parâmetros que pode ser utilizada para executar novamente o algoritmo múltiplas vezes, atualizando sucessivamente a BN à medida que se recolhem novas execuções. No entanto, a configuração ótima de parâmetros pode variar consideravelmente entre instâncias de um problema de um determinado domínio. Sendo assim, para lidar com isso, é construído este novo sistema que incorpora informação adicional sobre as instâncias do problema e utilizada o CBR como integrador da estrutura para as diferentes instâncias do mesmo problema, em que cada instância do problema lida com uma BN específica.

O CBR funciona através de um ciclo de *Retrieve, Reuse, Review* e *Retain*, ou seja, Recuperação, Reutilização, Revisão e Retenção de Casos, respetivamente. Cada caso na base de dados é representado como um par (problema, solução). A parte do problema contém a descrição de uma instância específica, enquanto a solução contém um modelo em forma de BN que estima a melhor configuração de parâmetros para o algoritmo. Na Recuperação, as características de uma nova instância do problema são utilizadas para indexar na base de dados e para recuperar a solução da BN potencialmente relevante. A Reutilização compreende a sugestão da melhor configuração de parâmetros para a instância atual a partir da solução da BN recuperada. Para este problema específico, para evitar soluções de baixa fiabilidade, são recomendadas k configurações mais prováveis. Na fase seguinte, a Revisão, o algoritmo é executado com cada configuração recomendada na fase anterior e reporta ao sistema o desempenho real obtido. Por fim, a Retenção consiste no registo/armazenamento de informação relativo ao processo executado, o que envolve as características do problema, a configuração de parâmetros e o respetivo desempenho obtido [274].

O sistema proposto foi testado e validado em Problemas de Design Geométrico *(Constraint-based Geometric Design Problem)*, especificamente para ajustar os parâmetros de um GA na

resolução de problemas de Identificação de Raízes *(Root Identiifcation Problem)*. A recolha e análise dos dados consiste na coleta de dados de execuções anteriores do algoritmo. Os valores dos parâmetros de controlo de um GA foram definidos consoante as recomendações existentes na literatura. Quando comparado com outros estudos, este sistema apresenta um desempenho quase igual (sem diferenças estatísticas significativas), mas consegue economizar uma quantidade significativa de tempo de computação. Um dos inconvenientes associados é que a construção da base de dados de casos iniciais requer um bom conhecimento e domínio por parte do utilizador e uma boa quantidade de experimentações, no entanto, é uma abordagem que se posiciona como uma excelente estratégia para ajustar automaticamente os parâmetros de um algoritmo, principalmente porque o faz para uma ampla variedade de problemas onde a configuração ótima de parâmetros depende da instância do problema [274].

2.10.2. Discussão e Caminhos Futuros

O uso de técnicas de ML tem-se mostrado promissor na resolução de Problemas de Escalonamento das Operações. A revisão da literatura indicou avanços significativos, especialmente no que concerne à sinergia entre diferentes tipos de aprendizagem e as técnicas tradicionais de otimização, como as Meta-Heurísticas. O principal objetivo nestas abordagens é aprimorar a parametrização e a eficiência das Meta-Heurísticas, resultando em soluções mais eficazes e adaptativas.

A definição dos valores dos parâmetros de uma Meta-Heurística pode ser realizada de forma *Offline* ou *Online*. Cada método de ajuste apresenta vantagens e desvantagens inerentes, tornando necessário analisar as características do processo de aplicação para determinar qual abordagem é mais vantajosa. A melhor definição dos parâmetros pode variar em função da instância do problema ou até mesmo em diferentes fases do processo de pesquisa na mesma instância. Nesses casos, torna-se crucial proceder a um ajuste dinâmico dos parâmetros.

A análise das publicações existentes na literatura revelou que as técnicas baseadas em RL se destacam e apresentam um progresso recente notável. Através de interações contínuas e trocas de feedback com um Ambiente, um Agente é capaz de aprender uma política que lhe permita optar por ações que maximizam a sua recompensa a longo prazo, conduzindo-o sucessivamente às melhores soluções.

Considerando os princípios subjacentes a um sistema baseado em RL, é possível adaptá-lo para englobar diferentes Meta-Heurísticas e aplicá-lo a vários tipos de problemas, desde o TSP até implantações como o *Job-Shop*, no entanto, ainda existem lacunas significativas em termos de exploração. Dada a eficácia dos mecanismos que incorporam RL, seria interessante ampliar o espectro de aplicações, implementando outras Meta-Heurísticas que ainda não têm sido alvo de um foco profundo de estudo, como o SA através do ajuste dos valores da TI e do CR, ou o TS, com a definição dinâmico do tamanho da Lista Tabu. Para as Meta-Heurísticas já exploradas, poderia ser interessante e vantajoso incidir sobre parâmetros que ainda não foram testados, pois, em vários casos, também são capazes de influenciar o desempenho do algoritmo.

Numa direção de investigação distinta, mas com potenciais níveis de complexidade e exigência superiores, surge a possibilidade de aplicação de outros tipos de Aprendizagem, como a

Aprendizagem Supervisionada e a Aprendizagem Não Supervisionada. Embora os algoritmos que constituem estes tipos de Aprendizagem apresentem resultados significativamente positivos noutros tipos de aplicações, as suas utilizações na parametrização de Meta-Heurísticas ainda carecem de uma exploração mais detalhada. Neste sentido, futuras pesquisas poderiam explorar a integração de técnicas híbridas que combinam diferentes métodos de ML, como, por exemplo, a utilização de Aprendizagem Supervisionada para identificar padrões em parâmetros ótimos a partir de históricos de execução, complementada por RL para ajustes dinâmicos durante a execução. Além disso, a aplicação de técnicas de Aprendizagem Não Supervisionada, como Técnicas de *Clustering*, poderia ajudar a categorizar diferentes instâncias de problemas e sugerir configurações de parâmetros mais adequadas para cada categoria. Desta forma, a combinação de várias técnicas de Aprendizagem pode superar as limitações inerentes a cada abordagem e proporcionar um ajuste de parâmetros mais robusto, adaptativo e capaz de encontrar soluções mais eficientes para uma ampla gama de Problemas de Otimização.

3. Protótipo de Parametrização de Meta-Heurísticas

Os Problemas de Otimização Combinatória, nos quais se incluem os Problemas de Escalonamento, possuem particularidades únicas que podem afetar diretamente a eficácia dos métodos de otimização utilizados. Selecionar os parâmetros que permitem a um método, como uma Meta-Heurística, obter o melhor desempenho numa dada instância de um problema é um processo tradicionalmente trabalhoso, mas extremamente importante [111].

O processo de parametrização de uma Meta-Heurística visa alcançar um equilibro entre a exploração e a diversidade. Os parâmetros influenciam diretamente a maneira como a técnica de otimização navega pelo espaço de soluções. Ajustes adequados podem garantir um balanço entre a procura por soluções ótimas e a exploração de novas regiões do espaço de soluções. Além disso, a eficiência computacional também é uma consideração essencial. Parâmetros bem otimizados contribuem para uma convergência mais rápida e para soluções de maior qualidade. Este aspeto é particularmente crítico em Problemas de Escalonamento [50, 121].

Considerando a complexidade e o trabalho inerentes ao processo de parametrização de Meta-Heurísticas, uma alternativa pode ser abordar esse processo utilizando técnicas de ML. Sendo assim, ao longo deste Capítulo 3 será apresentado o trabalho desenvolvido na formulação de um sistema completo capaz de parametrizar uma Meta-Heurística – o *Simulated Annealing*, utilizando RL para resolver o problema de Minimização do *Makespan* em *Job Shop*. Para além da descrição e caracterização do problema em si, também será apresentada a estrutura de todo o sistema, com destaque para o Sistema Inteligente de Parametrização, que é constituído por dois componentes: o Agente de Parametrização e o Ambiente de Otimização. Para o primeiro componente, será detalhado o processo de funcionamento do Agente, que descreve o modo como este é capaz de controlar o valor de cada parâmetro. Para o Ambiente de Otimização, serão detalhados os princípios que estão na base da criação de um ambiente de problemas em *Job Shop*, o que envolve a formalização do problema, dos estados e ações e do mecanismo de recompensa, e, para além disso, será abordado todo o processo de implementação do SA.

3.1. Problema de Escalonamento

No presente trabalho pretende-se analisar o desempenho de técnicas de otimização integradas com algoritmos de ML para resolver Problemas de Escalonamento. O problema utilizado no

estudo é um Problema de Minimização do *Makespan* em *Job-Shop* (JSSP, de *Job-Shop Scheduling Problem*), que atualmente representa as implantações industriais em vários setores.

O JSSP é um dos Problemas de Otimização Combinatória mais clássicos e relevantes que têm sido amplamente estudados. É um problema NP-Hard que pretende determinar a sequência ideal de atividades (ou tarefas) em cada máquina e pode ser descrito pelos seguintes aspetos:

→ Existe um conjunto finito de n atividades, $\mathrm{J} = \{\mathrm{J}_1, \mathrm{J}_2, \dots, \mathrm{J}_\mathrm{n}\}$ para serem processadas num conjunto finito de m máquinas, $\mathrm{M} = \{\mathrm{M}_1, \mathrm{M}_2, \dots, \mathrm{M}_\mathrm{m}\}$;

→ Cada atividade, $\mathrm{J_i}$, inclui uma sequência predefinida de h operações, $\mathrm{O} = \{\mathrm{O}_{\mathrm{i},1}, \mathrm{O}_{\mathrm{i},2}, \dots, \mathrm{O}_{\mathrm{i,h}}\}$. Cada operação deve ser executada numa determinada máquina e possui tempos de processamento especificados. $\mathrm{O}_{\mathrm{i,j}}$ representa a j-ésima operação $(\mathrm{j} = 1,2, \dots, \mathrm{h})$ da i-ésima atividade $(\mathrm{i} = 1,2, \dots, \mathrm{n})$, e $\mathrm{t}_{\mathrm{i,j}}$ é o respetivo tempo de processamento dessa operação, $\mathrm{O}_{\mathrm{i,j}}$.

Relacionados com as atividades e as máquinas, é importante mencionar alguns pressupostos que são os mais comuns e caracterizam um problema de Minimização do *Makespan* em *Job-Shop* [278]:

→ Cada máquina apenas pode processar uma operação de cada vez;

→ Cada operação apenas pode ser processada por uma máquina de cada vez;

→ Todas as atividades e máquinas estão disponíveis no momento inicial;

→ Não existem restrições de precedência entre as operações de diferentes atividades;

Na literatura existem diferentes formulações matemáticas para representar Problemas de Escalonamento em *Job-Shop*, que diferem em diversos parâmetros, como as variáveis utilizadas, o conjunto de restrições e os critérios de desempenho. Sendo assim, serão apresentados, de seguida, dois modelos que representam este tipo de problema.

Primeiro Modelo

O primeiro modelo apresentado é baseado nos estudos apresentados em [279, 280], e pode ser representado como $J_m \mid C_{max}$ através da representação de Graham. Para este caso, é importante mencionar a variável de decisão é $s_{i,j}$, que representa o tempo de início da operação $O_{i,j}$. Assim, como apresentado anteriormente na caracterização de um problema *Job-Shop*, existem n atividades que podem ser constituídas por h operações.

F.O

$$Min\ C_{máx} = Min\left\{Max \sum_{i=1}^{n} \sum_{j=1}^{h} \left(s_{i,j} + t_{i,j}\right)\right\} \tag{23}$$

S.A.

$$s_{i,j} + t_{i,j} \leq s_{i,j+1} \qquad ,1 \leq i \leq n; 1 \leq j \leq h-1 \qquad (24)$$

$$\left(s_{i,j} + t_{i,j} \leq s_{l,j}\right) ou \left(s_{l,j} + t_{l,j} \leq s_{i,j}\right) \qquad ,1 \leq i; l \leq n \leq; 1 \leq j \leq h \qquad (25)$$

$$s_{i,j} \geq 0 \qquad ,1 \leq i \leq n; 1 \leq j \leq h \qquad (26)$$

Neste primeiro modelo, o objetivo é determinar o tempo de início de cada atividade. A Função Objetivo, representada na Equação (23), é a minimização do tempo total de conclusão necessário para todos os trabalhos, conhecido na literatura como *makespan* [13, 279].

A Equação (24) é uma restrição que assegura o cumprimento da sequência de processamento de operações da mesma atividade, o que significa que o início do processamento da operação $O_{i,j+1}$ só acontece depois da operação imediatamente anterior, $O_{i,j}$, ter sido concluída. Esta restrição pode ser representada de diferentes maneiras. Muitos autores também utilizam $O_{i,j}$ e $O_{i,k}$ como duas operações consecutivas da atividade J_i, sendo $k > j$, ou seja, $(i,j) \rightarrow (i,k)$, e, substituem a Equação (24) por $s_{i,j} + t_{i,j} \leq s_{i,k}$, com $2 \leq k \leq h$.

Na Equação (25) define-se que cada máquina processa uma operação de cada vez. Neste caso $O_{i,j}$ *e* $O_{l,j}$ representam duas operações que necessitam de ser processadas na mesma máquina. Uma máquina apenas processa uma de cada vez porque $O_{l,j}$ só inicia o seu processamento após $O_{i,j}$ ter sido processada (ou $O_{i,j}$ depois de $O_{l,j}$, o que é equivalente porque não existem restrições de precedência entre operações de diferentes atividades). Por fim, a Equação (26) assegura que o tempo de início do processamento é igual ou superior a zero.

Este primeiro modelo é assim uma representação simples, mas completa do Problema de Minimização do *Makespan* em *Job-Shop*, que coloca em prática os principais pressupostos que caracterizam o problema.

Segundo Modelo

O segundo modelo baseia-se no estudo apresentado em [281] e difere-se do primeiro modelo porque inclui uma variável binária. As variáveis de decisão são apresentadas na Equação (27) e na Equação (28).

$$x_{i,j,k} = \begin{cases} 1, & se\ a\ atividade\ j\ \text{é}\ processada\ antes\ da\ atividade\ k\ na\ máq.\ i \\ 0, & caso\ contrário \end{cases} \qquad (27)$$

$$s_{i,j} = tempo\ de\ início\ da\ atividade\ j\ na\ máquina\ i \qquad (28)$$

Este modelo considera que $t_{i,j}$ representa o tempo de processamento da atividade j na máquina i. O número de atividades varia até n e o número de máquinas até m. Para representar a h-ésima operação da atividade j utiliza-se σ_h^j. Assim, para cada atividade j, é fornecida a lista $\left(\sigma_1^j, \dots, \sigma_h^j, \dots, \sigma_m^j\right)$ de máquinas com a ordem de processamento dessa mesma atividade.

F.O

$$Min\ C_{máx} \tag{29}$$

S.A.

$$s_{\sigma_{h-1}^{j},j} + t_{\sigma_{h-1}^{j},j} \leq s_{\sigma_{h}^{j},j} \qquad ,1 \leq j \leq n; 2 \leq h \leq m \tag{30}$$

$$s_{i,j} \geq s_{i,k} + t_{i,k} - M \times x_{i,j,k} \qquad ,1 \leq j; k \leq n; j \leq k; 1 \leq i \leq m \tag{31}$$

$$s_{i,k} \geq s_{i,j} + t_{i,j} - M\left(1 - x_{i,j,k}\right) \qquad ,1 \leq j; k \leq n; j \leq k; 1 \leq i \leq m \tag{32}$$

$$C_{máx} \geq s_{\sigma_{m}^{j},j} + t_{\sigma_{m}^{j},j} \qquad ,1 \leq j \leq n \tag{33}$$

$$x_{i,j,k} \in \{0,1\} \qquad ,1 \leq j; k \leq n; 1 \leq i \leq m \tag{34}$$

$$s_{i,j} \geq 0 \qquad ,1 \leq j \leq n; 1 \leq i \leq m \tag{35}$$

Este tipo de problema pretende determinar a sequência de processamento das atividades em cada máquina, bem como o momento de início de cada uma. É uma representação mais complexa que recorre a mais variáveis para incluir informação sobre as restrições que devem existir entre operações da mesma atividade e o que cada máquina deve estar a processar em cada momento.

A Equação (29) representa o objetivo do problema, que é a minimização do *makespan*, e relaciona-se com a Equação (33), no qual as bases do JSSP são definidas. Neste caso, é mencionado que o *makespan* deve ser, no mínimo, superior ao tempo de conclusão mais longo da última operação de todas as atividades. A Equação (30) garante que cada operação que constitui uma atividade é processada na ordem requerida, ou seja, assegura-se o cumprimento da sequência de processamento de operações. A Equação (31) e a Equação (32) estabelecem que não se pode processar duas atividades numa máquina ao mesmo tempo. Os valores da variável binária são definidos na Equação (34), enquanto a Equação (35) assegura que o tempo de início de cada atividade numa máquina é igual ou superior a 0.

Para modelar um problema clássico de *Job Shop*, nomeadamente instâncias representativas do mesmo, é comum utilizar um grafo disjuntivo, $G = (V, C \cup D)$, mais conhecido pelo seu termo em inglês *Disjunctive Graph*. Neste tipo de representação, V é o conjunto de vértices (ou nós) correspondentes às atividades. Existem dois vértices adicionais, que representam a origem/início e o fim do Escalonamento e possuem tempo de processamento nulo. C refere-se ao conjunto de arcos conjuntivos entre nós que compreendem operações da mesma atividade e estão orientados de modo a representar as relações de precedências. Por sua vez, D representa o conjunto de todas as arestas disjuntivas entre nós de operações que requerem a mesma máquina para serem processadas. Cada arco tem associado a si o tempo de processamento da atividade onde o arco começa [282, 283].

Um exemplo de uma instância de um problema de JSSP, com $n = 3$ e $m = 3$, isto é, 3 atividades, $J = \{J_1, J_2, J_3\}$, e 3 máquinas, $M = \{M_1, M_2, M_3\}$, é apresentado na Tabela 9. A representação do grafo disjuntivo é apresentada na Figura 19.

Tabela 9- Exemplo de uma instância de um problema 3x3 *Job Shop* (n=3, m=3) [284]

Atividade	**Máquina (Tempo de Processamento)**		
J_1	$1(7)$	$2(6)$	$3(6)$
J_2	$2(6)$	$3(7)$	$1(7)$
J_3	$3(8)$	$2(5)$	$1(9)$

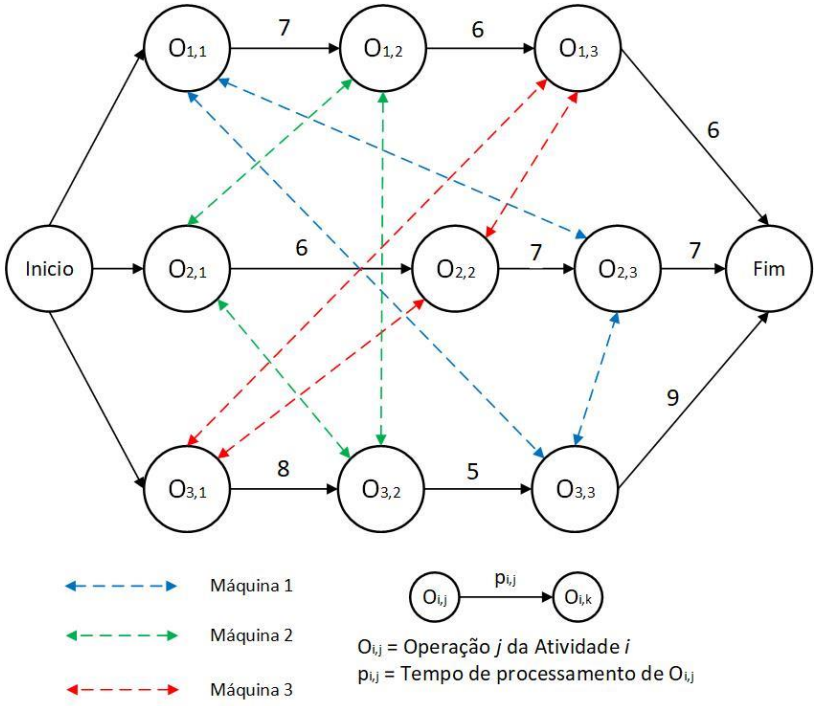


Figura 19- Grafo disjuntivo para a instância de um problema 3x3 *Job Shop*

O problema de minimização do *makespan* em *Job Shop*, $J_m|C_{max}$, é assim um tipo de problema com elevado interesse académico que tem sido alvo de um estudo intenso desde que foi introduzido, mas também assume uma extrema importância no contexto real. Desenvolver soluções capazes de o resolver é extremamente importante para superar os desafios relacionados com a produção e aumentar a eficiência operacional e a competitividade das empresas. Ao otimizar as medidas de desempenho associadas ao mesmo, consegue-se melhorar a capacidade de resposta à procura dos clientes e, consequentemente, melhorar a posição competitiva de uma organização.

3.2. *Framework* de Parametrização através de RL

O *Framework* de Parametrização tem como objetivo encontrar os parâmetros adequados para que uma Meta-Heurística tenha o melhor desempenho possível numa determinada instância de um Problema de Escalonamento com Minimização do *Makespan* em *Job-Shop*. O sistema

funciona como uma abordagem dinâmica e adaptativa porque incorpora RL no seu funcionamento, o que significa que existe um Agente que interage com um Ambiente e aprende a política "ótima" de parametrização. Para resolver um determinado problema, uma Meta-Heurística, em vez de ser parametrizada através de ajustes manuais ou do uso de outras Heurísticas, recorre a um Agente que utiliza os princípios que caracterizam o RL para explorar o espaço de parâmetros e descobrir as configurações de parâmetros que permitem obter as melhores soluções.

A estrutura geral do *Framework* desenvolvido neste trabalho é apresentada na Figura 20. Em primeiro lugar, é possível identificar o problema de Minimização do *Makespan* em *Job-Shop* e o Leitor de Instâncias. O cerne de todo este *Framework*, que consiste no sistema responsável por parametrizar uma Meta-Heurística e resolver o problema em si, é representado pelo Sistema Inteligente de Parametrização (SIP), da qual faz parte o Agente de Parametrização (AP) e o Ambiente de Otimização (AO). No seu interior, o AO possui a Meta-Heurística (MH) implementada. Após todo o processo de parametrização, o resultado corresponde à configuração ótima de parâmetros da MH e à respetiva solução do Problema de Escalonamento.

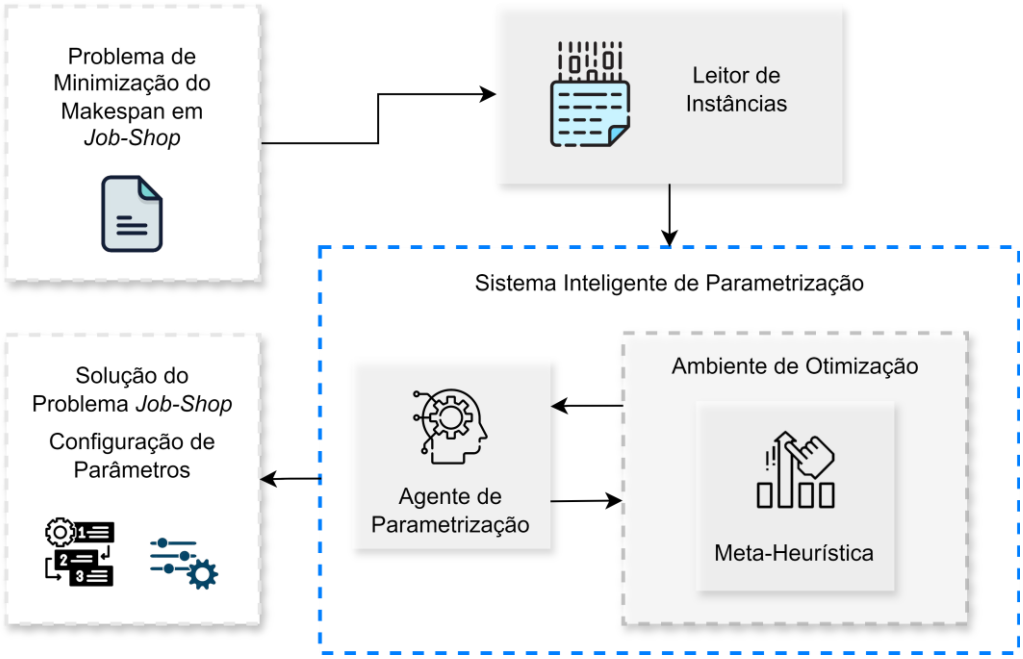


Figura 20- Diagrama representativo do *Framework* de Parametrização através de RL

A base para o funcionamento do *Framework* de Parametrização é o fornecimento de uma instância de um problema *Job-Shop*. O Leitor de Instâncias é responsável por transformar a informação presente no documento em estruturas de dados adequadas a usadas pelo SIP. A MH é responsável por resolver o problema *Job-Shop*, no entanto, necessita de informação que descreva o problema e dos valores para os parâmetros. As características relacionadas com as atividades, operações e máquinas estão definidas no Ambiente *Job-Shop*, enquanto o AP é quem define os valores dos parâmetros – representados como um "estado", e controla o modo como variam – através de ações. O AP comunica com o AO, que executa uma ação e aciona a execução da MH, que utiliza as respetivas configurações de parâmetros para explorar o espaço de soluções na procura por uma solução de alta qualidade.

Neste sistema, o SA é capaz de gerar soluções candidatas, avaliar as suas qualidades através de uma função de custo, e obter a melhor solução e o respetivo valor da função objetivo – *makespan*. Com base neste último valor, calcula-se a recompensa dentro do AO, que é recebida pelo AP. Com base neste feedback, o AP observa o impacto que cada configuração teve no objetivo do problema e decide como ajustar os parâmetros para melhorar o processo de pesquisa e alcançar a melhor solução possível para o problema em questão. Isto significa que toda a troca de informação entre o AP, o AO e a MH, isto é, a interação entre um Agente, um Ambiente que executa o SA, e o próprio módulo SA, é repetida até encontrar a melhor solução para o problema e a configuração de parâmetros correspondente.

O *Framework* de Parametrização desenvolvido neste trabalho é assim constituído por vários componentes que estão posicionados e agrupados em sistemas capazes de implementar e seguir os princípios de RL. Todos os constituintes identificados na descrição do fluxo de informação serão detalhados ao longo deste capítulo.

3.2.1. Leitor de Instâncias

O Leitor de Instâncias é responsável por descodificar a informação presente num ficheiro de texto, transformando-a em estruturas de dados utilizáveis pelo programa, que representam os vários elementos do problema, como o número de atividades, o número de máquinas e o tempo de processamento de cada operação em cada máquina.

Este componente é capaz de lidar com as instâncias nos formatos (ou especificações) disponibilizados nas bases de dados mais utilizadas em estudos de investigação, como a *OR-Library* [7]. Neste caso é importante destacar o "Formato Standard", que é a especificação mais utilizada, onde se incluem autores como Lawrence [285] e Adam, Balas e Zawack [286], e o "Formato Taillard" [287]. A Figura 21 apresenta um exemplo do formato de uma instância 20x5 no "Formato Standard". Na primeira linha, existem dois números, com o primeiro a corresponder ao número de atividades e o segundo ao número de máquinas. Para a instância exemplo em análise, o leitor observa que existem 20 atividades e 5 máquinas. De seguida, cada linha corresponde a uma atividade, e a informação, para cada operação, é apresentada com a indicação da máquina, seguida pelo tempo de processamento correspondente. A numeração das máquinas inicia-se em 0. Por exemplo, considerando a linha 2 que é referente à primeira atividade, a primeira operação é processada na máquina 2 com um tempo de processamento de 34 unidades de tempo, a segunda operação passa na máquina 1 e demora 21 unidades de tempo, e assim sucessivamente, com a última operação a ser executada na máquina 4 com um tempo de processamento de 95 unidades de tempo. A informação relativa a cada atividade é guardada pelo leitor numa variável indexada de duas dimensões, onde cada uma corresponde a uma operação e armazena informação relativa à máquina onde é processada e ao respetivo tempo de processamento.

```
20 5
2 34 1 21 0 53 3 55 4 95
0 21 3 52 1 71 4 16 2 26
0 12 1 42 2 31 4 98 3 39
2 66 3 77 4 79 0 55 1 77
0 83 4 37 3 34 1 19 2 64
4 79 2 43 0 92 3 62 1 54
0 93 4 77 2 87 1 87 3 69
4 83 3 24 1 41 2 38 0 60
4 25 1 49 0 44 2 98 3 17
0 96 1 75 2 43 4 77 3 79
0 95 3 76 1  7 4 28 2 35
4 10 2 95 0 61 1  9 3 35
1 91 2 59 4 59 0 46 3 16
2 27 1 52 4 43 0 28 3 50
4  9 0 87 3 41 2 39 1 45
1 54 0 20 4 43 3 14 2 71
4 33 1 28 3 26 0 78 2 37
1 89 0 33 2  8 3 66 4 42
4 84 0 69 2 94 1 74 3 27
4 81 2 45 1 78 3 69 0 96
```

Figura 21- Exemplo de uma instância do problema *Job-Shop*: Formato Standard [7]

3.2.2. Sistema Inteligente de Parametrização

A visão geral da estrutura do SIP é apresentada na Figura 22. É possível identificar a divisão em vários componentes: o Ambiente de Otimização - que incorpora a Meta-Heurística, e o Agente de Parametrização - na qual está implementado o algoritmo *Q-Learning*.

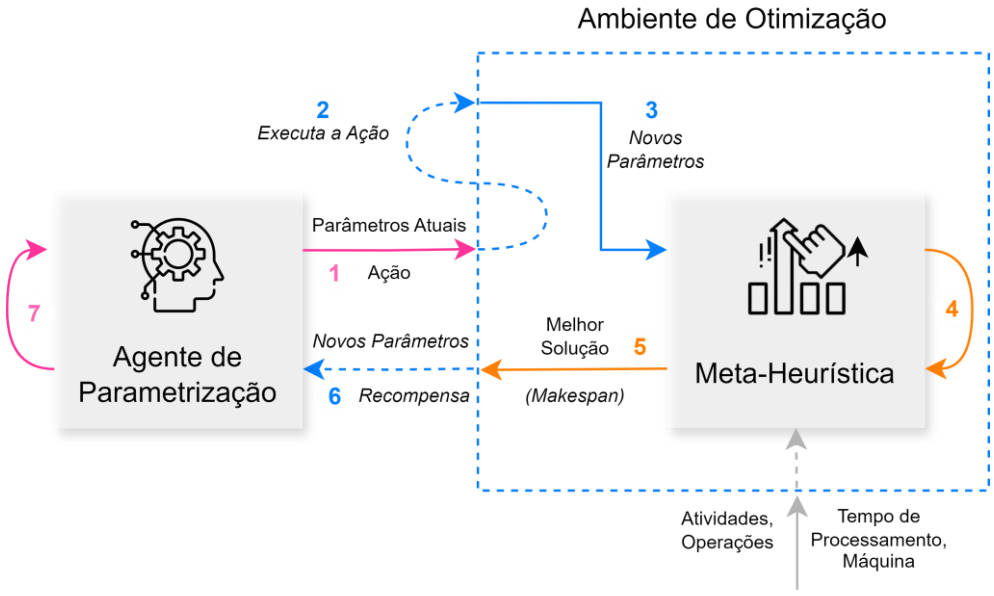


Figura 22- Fluxo de informação entre os componentes do AIP

O SIP está desenvolvido para, através da cooperação entre os seus diferentes componentes, definir a melhor configuração de parâmetros capaz de permitir a obtenção de uma boa solução. Num determinado momento, o AP é responsável por provocar uma alteração nos valores dos parâmetros atuais, o que é feito através de uma determinada ação (1). O AO é o local onde as decisões do AP são executadas. Assim, esta ação provoca uma alteração nos valores dos parâmetros a parametrizar (2). Para os testar, o AO executa a MH (3), que, ao longo de várias iterações, gera a melhor solução e o *makespan* correspondente (4), retorna-os para o AO (5)

que, com base nesse valor, calcula a recompensa e envia essa informação para o AP (6). O componente de aprendizagem do AP, no qual está implementado o *Q-Learning*, é responsável por seguir os princípios que caracterizam o algoritmo (atualização da tabela Q), decidir que alterações deve realizar – política *epsilon-greedy* na escolha da ação (7), e iniciar novamente este ciclo até considerar que encontrou as melhores soluções para as variáveis em questão.

A interação entre cada componente que constitui o SIP, nomeadamente relativamente à informação enviada (ou recebida), é determinante para o sucesso de todo o *Framework*. Este sistema, que funciona através de um Agente que interage com o Ambiente, explora o espaço de parâmetros da MH, experimenta diferentes configurações de parâmetros para entender como elas afetam o desempenho do algoritmo, e, com base nesse feedback, seleciona uma ação a ser executada, o que significa que é assim um sistema que está desenvolvido para proporcionar as condições necessárias à aprendizagem de uma política de parametrização que permita obter as melhores soluções.

Os princípios que são a base do funcionamento do *Framework* de Parametrização e caracterizam todo o processo de RL, serão detalhados, de seguida, de forma pormenorizada.

Ambiente de Otimização – Formalização do Ambiente, da Recompensa e das Ações

No RL, existem dois elementos que são a base de todo o processo: a Entidade, que toma as decisões, e o Ambiente, que é o local onde este navega e todas as decisões são executadas. Considerando a abordagem típica de um problema desta área, é fundamental modelar o problema a executar, o que engloba a definição de um conjunto de estados, S, de ações possíveis, A, e da fórmula ou função que descreve a recompensa, R.

No contexto do problema a resolver neste trabalho, o Ambiente necessita de incluir informação sobre as principais características que descrevem o problema, a técnica de otimização que o resolve, e os respetivos parâmetros. As principais características do problema – nomeadamente número de atividades, número de operações por atividade, tempos de processamento e máquina onde cada operação é processada, são recebidas, pelo Ambiente, provenientes do LI, que organiza a informação em estruturas de dados prontas para serem posteriormente enviadas e utilizadas pela MH.

Relativamente aos parâmetros, é crucial abordar e entender o tipo de variáveis que estes podem assumir, porque isso influencia o comportamento e o desempenho da Meta-Heurística. Os parâmetros podem ser classificados como variáveis quantitativas – quando assumem valores numéricos que fazem sentido e podem ser medidos numa escala quantitativa, ou qualitativas, também conhecidas como variáveis categóricas – quando o parâmetro não usa valores numéricos, mas descreve a informação por categorias. No âmbito das variáveis quantitativas, um parâmetro é classificado como variável contínua se poder assumir um número infinito de valores dentro de um intervalo específico, e como variável discreta, quando apenas pode ter um conjunto finito ou enumerável de valores específicos.

Na maioria dos parâmetros que caracterizam uma Meta-Heurística e que podem ser considerados variáveis quantitativas, a decisão entre tratá-los como variáveis contínuas ou discretas pode influenciar diretamente a eficiência do *Framework* de Parametrização. Em

teoria, variáveis contínuas têm uma precisão "infinita", no entanto, na prática computacional, esta precisão é limitada devido ao número finito de bits disponíveis para armazenar a informação. Apesar de permitirem a representação de uma ampla gama de valores e serem uma boa opção para parâmetros que exigem ajustes finos e contínuos, podem ter como consequência um aumento do esforço computacional. Por sua vez, em várias situações, as variáveis discretas são preferíveis pela sua simplicidade e eficiência computacional, uma vez que, em teoria, são mais fáceis de manipular. No contexto em que se insere este trabalho, durante o processo de aprendizagem, é fundamental proporcionar ao Agente a exploração dos vários estados e das diferentes ações em cada estado. Para cada Meta-Heurística, é fundamental compreender a influência de cada parâmetro e definir com atenção os seus limites e os valores que pode assumir. Uma representação incorreta de variáveis pode traduzir-se numa elevada quantidade de dados a armazenar desnecessariamente, num aumento do número de estados a explorar por parte do Agente, dificultando assim o seu processo de aprendizagem, e numa dificuldade acrescida de manipulação de dados, o que aumenta o esforço computacional requerido.

Tendo em consideração que o objetivo é resolver um problema *Job-Shop* através da Meta-Heurística, e os parâmetros da mesma são escolhidos por um Agente, isto significa que um estado pode ser definido como uma configuração de parâmetros, e o conjunto de estados, S, contém todas as configurações possíveis. Com isto, é possível estabelecer que uma mudança de estado acontece quando existe uma alteração no valor de pelo menos um dos parâmetros.

De forma genérica, considerando uma Meta-Heurística caracterizada por quatro parâmetros – $P1, P2, P3\ e\ P4$, a Equação (36) apresenta a representação de um estado, s, no instante de tempo t, no *Framework* de Parametrização desenvolvido neste trabalho. Qualquer parâmetro pode apresentar um valor contínuo, discreto ou categórico, o que significa que respeitam um determinado intervalo ou uma lista de várias categorias.

$$s_t = (P1, P2, P3, P4) \tag{36}$$

Em cada momento, o Agente tem de tomar uma ação, o que significa alterar pelo menos o valor ou a palavra associada a cada parâmetro para mudar de estado. Caso o parâmetro seja uma variável quantitativa, o Agente pode decidir aumentar, manter ou diminuir o seu valor. A quantidade alterada no valor já depende especificamente do tipo de parâmetro, ou seja, da sua natureza – contínua ou discreta. Na situação onde é uma variável categórica, pode ser escolhida uma categoria entre as que estão disponíveis, o que significa que se pode manter a palavra representativa ou alterar para outra. Considerando, para efeitos de exemplo ilustrativo, os parâmetros $P1\ e\ P2$ como variáveis contínuas ou discretas, e $P3\ e\ P4$ como variáveis categóricas, a Equação (45) apresenta as possíveis ações para $P1\ e\ P2$ e a Equação (38) para $P3\ e\ P4$, onde CTG é representativo de uma categoria. O conjunto de Ações, A, engloba assim todas as combinações que podem ser feitas com os parâmetros a parametrizar pelo Agente.

$$a_{P1} = a_{P2} = [\text{'}UP\text{'}, \text{'}KEEP\text{'}, \text{'}DOWN\text{'}] \tag{37}$$

$$a_{P3} = a_{P4} = [\text{'}CTG1\text{'}, \text{'}CTG2\text{'}, \text{'}CTG3\text{'}, \text{'}CTG4\text{'}] \tag{38}$$

Considerando o funcionamento do SIP, num determinado momento, uma ação tomada pelo Agente provoca uma transição de um estado específico para outro estado. Para testar esta transição, que significa perceber o desempenho de uma nova configuração de parâmetros, o AO executa a MH, recebe o valor do *makespan* e calcula a recompensa que o Agente receberá. De forma geral, a função de recompensa R é calculada através da Equação (39), onde $C_{máx}$ representa o *makespan*.

$$R_{s+1} = C_{máx_s} - C_{máx_{s+1}} \tag{39}$$

De facto, a Equação (39) recompensa o Agente quando uma ação resultou numa melhoria do *makespan*, e penaliza-o quando conduziu a valores maiores, no entanto, é crucial desenvolver uma função completa que, para além disso, também seja capaz de o incentivar a explorar as combinações de parâmetros que conduzam ao melhor *makespan* porque o seu objetivo, para um determinado problema, é identificar essa configuração. Sendo assim, em qualquer episódio, para qualquer ação que seja capaz de melhorar o *makespan*, e esse seja o melhor *makespan* encontrado até ao momento, ou seja, naquele instante é o maior candidato a ser a solução final para o objetivo do Agente, é adicionada a quantidade FR (Fator de Recompensa) à função de recompensa da Equação (39), como é apresentado na Equação (40). Este Fator de Recompensa pode ser adaptado às características da instância e do tipo de problema, mas deve apresentar uma ordem de grandeza superior às recompensas normais calculadas através da Equação (39).

$$R_{s+1} = C_{máx_s} - C_{máx_{s+1}} + FR \tag{40}$$

Agente de Parametrização – Agente *Q-Learning*

O Agente desempenha um papel fundamental no SIP porque atua como o componente responsável por tomar decisões para ajustar os parâmetros da técnica de otimização. Através da aplicação dos princípios que caracterizam o *Q-Learning*, o principal objetivo do Agente é aprender a melhor política de seleção de ações com base nas interações com o Ambiente, nas informações que recebe do mesmo e nas experiências de treino. Neste caso, o uso de uma tabela de valores Q, onde cada entrada representa o valor esperado da recompensa futura que o Agente pode obter ao tomar uma determinada ação num determinado estado, e a constante interação com o Ambiente, permitem que o Agente tome decisões mais informadas e eficazes, uma vez que os valores Q tendem a convergir para estimativas mais corretas de valor. Em cada iteração do processo de treino, o valor de Q é atualizado, o que permite ao Agente saber quais são as ações que são capazes de provocar melhorias mais significativas, guiando assim o seu processo de exploração.

A seleção de ações é influenciada não apenas pela qualidade das soluções obtidas durante a resolução da Meta-Heurística, mas também pelo equilíbrio entre *exploration* e *exploitation* durante o processo de aprendizagem do Agente. Para balancear estes dois conceitos segue-se a estratégia de seleção *epsilon-greedy* [189, 195]. Neste caso, podem ser implementadas diversas abordagens, sendo uma considerada como uma das mais comuns, iniciar com um valor de ε mais elevado, e depois reduzi-lo ao longo do tempo, à medida que o Agente ganha mais

experiência e confiança nas suas estimativas. Esta redução pode ser realizada de diferentes maneiras e permite que o Agente explore amplamente no início, e, à medida que o processo de aprendizagem avança, concentre-se em ações que pareçam ser as mais promissoras.

3.3. Protótipo do *Framework* de Parametrização

O Protótipo do *Framework* de Parametrização foi implementado em *Python*, no Microsoft Visual Code (versão 1.93.0), seguindo os princípios de cada componente do *Framework* enunciados na Secção 3.2. A preferência por esta linguagem de programação deve-se ao facto de ser considerada uma das mais utilizadas em projetos que envolvam ML e/ou trabalho com análise e tratamento de dados, e à facilidade e rapidez em desenvolver projetos devido a toda a informação existente disponível.

O objetivo foi implementar uma Meta-Heurística de uso e estudo frequente, onde os parâmetros que a caracterizam possam influenciar significativamente o seu desempenho, aumentando assim a importância da parametrização e o interesse para perceber como é que uma área emergente, como o RL, pode ter um impacto positivo neste algoritmo. Tendo isto em consideração, implementou-se o *Simulated Annealing*, que é então o componente responsável por resolver o problema de Minimização do *Makespa*n em *Job-Shop*.

A estrutura que permite implementar o SA e colocar em prática os princípios abordados na Secção 2.5.1., nomeadamente o algoritmo apresentado na Tabela 4, é apresentada na Figura 23, onde é possível identificar a existência de três módulos: o *SA()*, o *Neighborhood()* e o *Fitness()*. Este sistema, como um todo, é responsável por resolver efetivamente o problema em questão e funciona como um avaliador do desempenho da parametrização realizada por todos os componentes que constituem o AP.

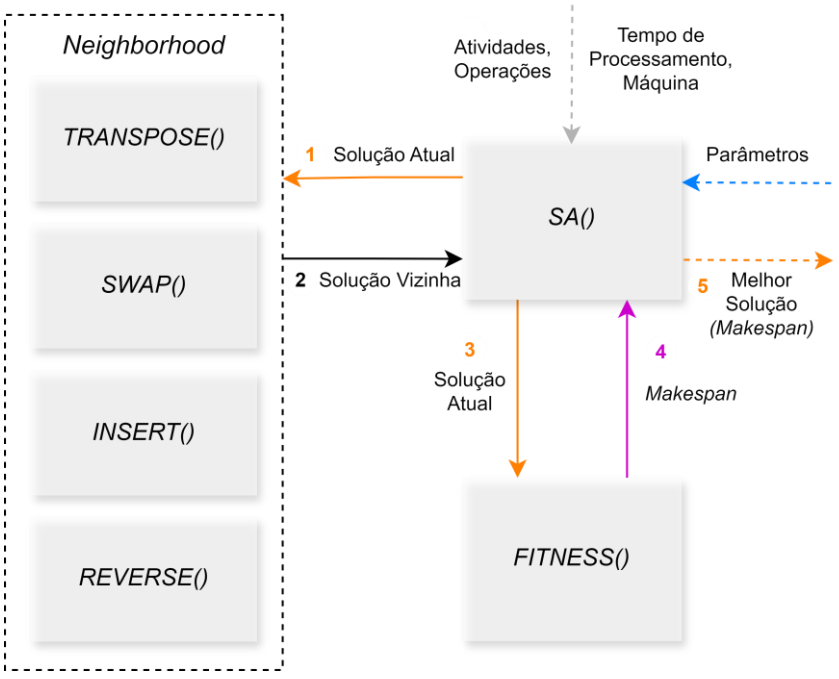


Figura 23- Estrutura de cada módulo que constitui o SA

Os três módulos interagem entre si e formam o algoritmo SA. O módulo *SA()* corresponde ao arquivo principal que lida com a leitura dos dados da instância do problema fornecida e da configuração de parâmetros, e executa o algoritmo SA, com base nos princípios do algoritmo apresentado na Tabela 4 e através da interação com os outros dois módulos. É neste módulo que se coloca em prática uma das particularidades que diferencia este algoritmo – em cada iteração, aceita soluções piores com uma determinada probabilidade, que depende da Temperatura Atual, T, e da diferença entre a função objetivo da solução candidata e da solução atual – Equação (4). Para manipular as soluções, ou seja, gerar novos vizinhos, o módulo *SA()* recorre ao *Neighborhood()*, onde estão implementados diferentes métodos de estrutura de vizinhança – *Transpose, Swap, Insert* e *Reverse*. Por sua vez, a avaliação de soluções, em termos de *makespan*, é realizada pelo módulo *Fitness()*, que fornece assim uma métrica objetiva para avaliar a qualidade das soluções geradas pelo SA, permitindo a comparação e seleção das melhores soluções.

3.3.1. *Simulated Annealing* - Formalização do Ambiente e das Ações

No caso específico do SA, a Meta-Heurística utilizada para resolver um problema de Minimização do *Makespan* em *Job-Shop*, os parâmetros que o caracterizam são a Temperatura Inicial, o Esquema de Arrefecimento e o *Epoch Length*. A adicionar a isto, existem parâmetros comuns com várias Meta-Heurísticas, como a Estrutura de Vizinhança e o Critério de Interrupção [50, 82].

A literatura refere que podem ser utilizados vários Esquemas de Arrefecimento, que descrevem o modo como a Temperatura T diminui ao longo da execução do algoritmo. O Protótipo do *Framework* de Parametrização possui o Esquema Geométrico implementado, que é representado pela Equação (6), onde α – o *Cooling Rate*, é o parâmetro a parametrizar. Para o Critério de Interrupção ou Paragem, o algoritmo pode ser interrompido quando a Temperatura atinge um valor mínimo, definido em função do problema, quando se atinge um determinado número de iterações ou quando se verifica uma certa quantidade de iterações onde a solução avaliada não melhora. De forma a avaliar sempre o mesmo número de soluções em cada execução do SA, no Protótipo utiliza-se o critério do número de iterações, mas este é definido previamente, de acordo com valores recomendados, para permitir que se avalie sempre o mesmo número de soluções em cada execução do SA [288].

Tendo assim em consideração o que caracteriza o SA, no Protótipo do *Framework* de Parametrização, um estado é representado por quatro parâmetros: Temperatura Inicial (TI), *Cooling Rate* (CR), *Epoch Length* (EL) e Estrutura de Vizinhança (EV). A Equação (41) apresenta a representação de um estado, s, no instante de tempo t. É importante mencionar que a TI, o CR e o EL representam variáveis discretas, enquanto a EV é uma variável categórica.

$$s_t = (TI, CR, EL, \text{`}EV\text{'}) \tag{41}$$

Em cada momento, a ação que o Agente pode tomar depende diretamente do tipo de parâmetro. Neste aspeto é importante mencionar dois aspetos – a definição do intervalo de valores ou das categorias disponíveis para cada parâmetro, e o modo como cada parâmetro

pode variar ou ser alterado, isto é, que valores pode ter dentro de cada intervalo. Todas estas definições são fundamentais para o sucesso de todo o Protótipo do *Framework* de Parametrização porque influenciam o comportamento do Agente e, consequentemente, a capacidade de o SA explorar o espaço de soluções de forma eficaz.

Como um estado é constituído por quatro parâmetros, e para cada parâmetro existem pelo menos três opções possíveis, é importante não utilizar intervalos de valores grandes para não criar uma tabela de valores q de elevada dimensão e não ser necessário armazenar uma grande quantidade de dados. Simultaneamente, também é fundamental não restringir muito os valores e criar intervalos muitos pequenos, porque isso pode limitar os processos de exploração e de aprendizagem do Agente.

O *Framework* de Parametrização está desenvolvido para definir o valor de cada parâmetro em função das características do problema a resolver. Para a TI, é implementada a métrica proposta em [156], no qual este parâmetro é determinado em função de uma determinada probabilidade de aceitação recomendada. Sendo assim, para definir os níveis da TI, são amostradas aleatoriamente 5000 soluções do SA, e a TI é obtida, com base na Equação (10), mas considerando a média da amostra, para probabilidades de aceitação de 0.8 e 0.9, que correspondem aos limites inferior e superior do intervalo, respetivamente.

Relativamente ao Esquema de Arrefecimento, foi implementado o Esquema Geométrico, um dos mais utilizados na literatura referente ao SA, onde o valor da Temperatura é multiplicado por uma taxa de arrefecimento α [289]. Para o valor de α, as recomendações de diversos autores variam nos limites do intervalo, nomeadamente no limite inferior. Neste caso, no Protótipo, α pode assumir qualquer valor compreendido no intervalo $[0.80, 0.99]$ [290], o que permite assim adotar valores recomendados na literatura e, ao mesmo tempo, não definir um intervalo com uma dimensão considerada elevada para evitar tornar o processo de aprendizagem do Agente mais difícil.

No caso do EL, que define o número de iterações, L, executadas a uma determinada Temperatura T, é comum utilizar um valor representativo da multiplicação de uma constante pelo número de vizinhos, ou, como é mencionado em [79, 163], este parâmetro deve ser proporcional ao tamanho do problema. Para o primeiro caso, o valor de L varia em função da EV utilizada, logo seria necessário criar um intervalo de valores que abrangesse todas as EV's, o que, para além de aumentar o tempo de execução, também permite a possibilidade de, para uma determinada EV, ter valores de L completamente desajustados. Sendo assim, é implementado o critério da dimensão do problema. Considerando o tipo de problema a resolver – Minimização do *Makespan* em *Job-Shop*, o *Epoch Length*, L, resulta da multiplicação entre o número de atividades e o número de máquinas, ou seja, $m \times n$. Para definir os limites superior e inferior do intervalo considera-se uma variação de 5% do valor de L. Portanto, o limite inferior é dado por $L \times 0.95$ e o limite superior por $L \times 1.05$. Relativamente à Estrutura de Vizinhança, definiu-se que, a partir de uma solução, as soluções vizinhas podem ser geradas através das estruturas mais comuns, aplicadas a diversos problemas - *Swap*, *Insert, Transpose* e *Reverse*.

Para os primeiros três parâmetros mencionados – TI, CR e EL, também é necessário definir que valores cada um pode tomar dentro de cada intervalo, uma vez que se podem assumir como

variáveis discretas. No caso dos parâmetros TI e EL, estes apenas podem ser números inteiros, enquanto o CR pode apresentar um valor com duas casas decimais. Para a EV, a palavra associada ao parâmetro é uma entre as quatro disponíveis.

Considerando assim o tipo de valor associado a cada parâmetro, o Agente pode decidir, em cada momento, aumentar, manter ou diminuir os valores da TI, do CR e do EL, como é apresentado na Equação (42). Para a TI, quando existe alteração no valor, esta é de cem unidades, para o CL é de uma centésima, enquanto para o EL é de uma unidade. No caso da EV, uma ação significa alterar a estrutura que o SA deve utilizar para gerar vizinhos, entre as quatro estruturas disponíveis e apresentadas na Equação (43). No Protótipo do *Framework* de Parametrização, o conjunto de Ações, A, engloba assim todas as combinações possíveis que podem ser feitas com estes quatro parâmetros, o que corresponde a 108 combinações.

$$a_{TI} = a_{CR} = a_{EL} = [\text{'}UP\text{'}, \text{'}KEEP\text{'}, \text{'}DOWN\text{'}] \tag{42}$$

$$a_{EV} = [\text{'}TRANSPOSE\text{'}, \text{'}SWAP\text{'}, \text{'}INSERT\text{'}, \text{'}REVERSE\text{'}] \tag{43}$$

As ações do Agente afetam o Ambiente. Quando o Agente executa uma ação, o Ambiente recebe-a como dado de entrada e adapta a sua configuração interna como uma reação, passando para um novo estado. No instante de tempo t do estado s, quando o Agente toma, por exemplo, a ação a – Equação (44), s_{t+1} representa o novo estado, como é apresentado na Equação (45).

$$a_t = [\text{'}DOWN, UP, DOWN, SWAP\text{'}] \tag{44}$$

$$s_t = [2000, 0.81, 8, \text{'}TRANSPOSE\text{'}] \longrightarrow s_{t+1} = [1999, 0.81, 7, \text{'}SWAP\text{'}] \tag{45}$$

Relacionado com o SA, também é fundamental abordar o modo de representação de uma solução, que, neste contexto, corresponde a um programa de Escalonamento. Neste caso, recorre-se à Codificação de Permutação com o objetivo de definir um método que permita explorar o espaço de pesquisa de soluções e interpretá-lo de forma completa. Considerando uma determinada instância $n \times m$ *Job-Sho*p, no qual existem n atividades e m máquinas, cada solução é representada por um conjunto de números inteiros, compreendidos entre 1 e n, sendo cada valor repetido m vezes. Isto significa que o valor de cada elemento representa a atividade com o mesmo número de identificação, e o número da operação é determinado pelo número de aparições do valor nos elementos situados antes dele. Por exemplo, na Figura 24, dada a solução [2 3 2 1 3 2 3 1 1], significa que existem três atividades, uma vez que $\{1,2,3\}$ representa $\{J_1, J_2, J_3\}$, e cada atividade possui três operações, o que faz com que cada valor da atividade seja repetido 3 vezes. O primeiro número 2 representa a primeira operação da segunda atividade a ser processada na máquina correspondente $(O_{2,1})$, o segundo número dois refere-se à segunda operação da segunda atividade, $(O_{2,2})$,, e assim sucessivamente. A solução [2 3 2 1 3 2 3 1 1] pode ser assim traduzida em $[O_{2,1}, O_{3,1}, O_{2,2}, O_{1,1}, O_{3,2}, O_{2,3}, O_{3,3}, O_{1,2}, O_{1,3}]$, onde $O_{i,j}$ representa a j-ésima operação da i-ésima atividade.

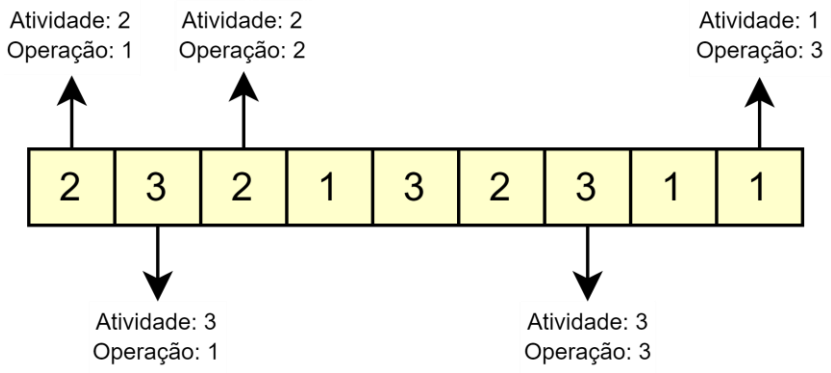


Figura 24- Exemplo da representação de uma solução

3.3.2. Agente de Parametrização – Treino e Parâmetros do algoritmo de RL

No método de resolução do Problema de Escalonamento que constitui a abordagem apresentada neste trabalho, ou seja, o processo de treino em si, este é iniciado a partir de um estado que resulta de uma configuração de parâmetros. Tendo em consideração os princípios que caracterizam o RL, é fundamental, para o correto funcionamento de todo o sistema, que o Agente inicie no mesmo estado para, a partir daí, interagir continuamente com o Ambiente e aprender, a partir dele, a política ótima que permita definir a configuração de parâmetros capaz de obter a melhor solução. Sendo assim, para os parâmetros TI, CR e EL, o estado inicial corresponde ao valor médio de cada intervalo. Para a EV, definiu-se a Estrutura *Transpose*, por ser uma das mais comuns em contexto de Escalonamento, uma vez que não existe nenhum motivo que justifique optar por alguma das estruturas disponíveis.

É importante mencionar que, sempre que necessário, tanto os valores dos limites dos intervalos de cada parâmetro, bem como o valor médio que corresponde ao estado inicial, foram ajustados para cada instância. Para a TI teve-se em consideração a quantidade correspondente à alteração do valor provocado por uma ação do Agente – cem unidades, o que significa que os limites inferior e superior foram arredondados para o número inteiro mais próximo e ajustados a múltiplos de cem, possibilitando assim obter um valor médio válido. No caso do CR, o valor médio é arredondado por excesso. Por sua vez, no EL, a variação de 5% é arredondada para o número inteiro mais próximo.

O processo de treino do Agente é modelado de forma a permitir que o mesmo seja capaz de identificar a calibração de parâmetros que permita ao SA obter a melhor solução para uma determinada instância. Relacionado com este processo destacam-se dois pontos importante: a definição da estratégia de avaliação de uma configuração de parâmetros e o critério de interrupção de um episódio de treino. O primeiro caso relaciona-se com o número de execuções do SA. Devido à natureza estocástica das Meta-Heurísticas, uma má parametrização pode resultar num bom desempenho da Meta-Heurística. Esta técnica incorpora diversos componentes aleatórios nos processos de gerar e analisar soluções, o que significa que, para uma determinada configuração de parâmetros, nem sempre se obtém a mesma solução. Sendo assim, são executadas 5 corridas da Meta-Heurística para cada configuração, e é utilizada a média dos três melhores resultados como solução final, nomeadamente no que diz respeito ao

makespan. Este aumento significativo do número de execução do SA tem como consequência um aumento do tempo computacional, no entanto, ao mesmo tempo, coloca-se em prática uma estratégia que influenciará positivamente o processo de aprendizagem do Agente porque tem como objetivo reduzir a variação de resultados devido à aleatoriedade existente.

Relativamente ao critério de interrupção de um episódio, de forma geral, um episódio termina quando o Agente (em conjunto com o SA) encontra uma solução melhor do que a melhor solução obtida nos episódios anteriores, ou quando alcançar um determinado número de iterações definido em função da cada instância. Numa fase inicial do treino, o valor de *epsilon* ainda é elevado, o que significa que existe uma exploração intensa, com o Agente a explorar diversos estados, várias ações, logo, durante este processo, pode ser encontrada a melhor solução para um determinado problema. Para impedir que, caso isso aconteça, os restantes episódios apenas terminem quando o número limite de iterações seja atingido, o que significaria, um aumento significativo e desnecessário do tempo computacional, é realizada, desde o início, uma contagem do número de episódios consecutivos no qual não se verificou uma melhoria na melhor solução. Se este valor ultrapassar um valor limite, o critério de interrupção sofre uma alteração: neste caso, todos os episódios seguintes terminam se essa melhor solução for encontrada, ou, caso não seja, aplica-se o número de iterações limite. Para cada instância, o número limite foi definido como sendo entre 8 e 10% do número de episódio, com exceção de algumas instâncias que serão explicadas durante o estudo computacional. A definição de um intervalo em vez de um único valor teve como objetivo obter um número limite inteiro, e permitir que instâncias com dimensões diferentes possuam valores limites diferentes porque o número de episódios também é distinto.

Relacionado com este aspeto, pode-se mencionar a função de recompensa, R. Nos princípios que estão na base do funcionamento do Protótipo do *Framework* de Parametrização, é mencionado que é necessário adicionar um Fator de Recompensa, FR, para realmente aumentar o valor da ação que permitiu obter, naquele instante, a melhor solução até ao momento. Neste Protótipo, o FR é calculado através da Equação (46), no qual $NMMS$ representa o Número de Melhorias na Melhor Solução encontrada, ou seja, é um fator que aumenta sempre que um episódio termina porque o sistema foi capaz de encontrar uma solução melhor. Tendo em consideração que esta solução pode ser realmente a melhor de todas, sempre que, nos episódios seguintes, o sistema for capaz de o encontrar, também se aplica a Função de Recompensa, apresentada na Equação (47).

$$FR = \left(\frac{NMMS}{2}\right) \times 100 \tag{46}$$

$$R_{s+1} = C_{máx_s} - C_{máx_{s+1}} + \left(\frac{NMMS}{2}\right) \times 100 \tag{47}$$

O objetivo desta Função de Recompensa não é só recompensar quando o Agente realmente encontra uma solução que é a melhor até ao momento, mas também fazê-lo quando, nas iterações seguintes, ele é capaz de a encontrar novamente. Com a atualização constante do $NMMS$, consegue-se priorizar progressivamente as melhores soluções, e todas as ações que

conduziram à última melhor solução encontrada são as que naturalmente apresentam o maior FR, logo serão as que possivelmente o Agente mais procurará na fase onde o valor de *epsilon* já é significativamente baixo.

Ao longo do processo de aprendizagem, as ações tomadas pelo Agente a cada mudança de estado são influenciadas pelos valores dos parâmetros que o definem. Neste caso, sendo a técnica de aprendizagem o *Q-Learning*, a Tabela 10 apresenta os valores do fator de desconto, γ, e do fator de aprendizagem, α. A definição de cada valor baseia-se na informação existente na literatura sobre a influência de cada parâmetro, mas também incorporou um processo de experiências de afinamento para permitir ao Agente obter os melhores resultados para o projeto em questão. Para o fator de desconto, γ, pretende-se que o Agente valorize recompensas futuras. As ações tomadas em cada momento têm consequências de longo prazo, portanto, um valor elevado, próximo da unidade, incentiva o Agente a procurar combinações de parâmetros que não sejam apenas eficazes a curto prazo, mas que também conduzam às melhores soluções no longo prazo. Quanto ao fator de aprendizagem, α, um valor significativamente baixo possibilita uma aprendizagem estável e evita grandes oscilações devido a mudanças rápidas nos valores de Q.

Tabela 10- Valores dos parâmetros para o algoritmo de RL

Parâmetro	**Valor**
Fator de Desconto, γ	0.99
Fator de Aprendizagem, α	0.10

Para o caso do valor do *epsilon*, pretende-se abordar o dilema *exploration vs exploitation* de modo a alcançar um equilíbrio entre os dois conceitos. A abordagem proposta neste trabalho divide o processo de treino em três fases distintas:

→ **1ª Fase:** O processo inicia-se com um valor de ε elevado. Numa fase inicial, um episódio termina sempre que o Agente é capaz de encontrar uma solução melhor, e adiciona-se o FR para lhe mostrar quais são as ações que conduzem a essas soluções. Como o conhecimento do Agente é inexistente, optar por valores de ε baixos nesta fase podia limitar o seu processo de exploração, e conduzi-lo rapidamente para ações que conduziram à melhor solução naquele momento, mas que ainda não é a melhor solução do problema ou instância. Sendo assim, a 1ª fase termina quando o valor limite de episódios sem melhoria da melhor solução é alcançado, e o valor de ε é constante e igual a 0.750;

→ **2ª Fase:** A fase intermédia termina quando se atinge 60% do número de episódios definido para o treino do Agente, e pode ser subdividida em quatro fases, todas com o mesmo número de episódios, sendo o valor do ε a única diferença entre elas: na primeira é 0.450, na segunda 0.200, na terceira 0.060 e na última 0.015. Esta redução constante simula uma redução progressiva com o objetivo de permitir que o Agente aperfeiçoe o conhecimento adquirido antes de iniciar a última fase;

→ **3ª Fase:** A última fase tem como objetivo aplicar o efeito de aprendizagem do Agente, ou seja, analisar se foi capaz de adquirir conhecimento suficiente que lhe permite identificar a melhor configuração de parâmetros repetidamente e, de preferência, num número de iterações consideravelmente inferior ao das fases anteriores. Esta fase corresponde aos últimos 40% do número total de episódios, e o valor de ε é constante e igual a 0.001.

Com a abordagem apresentada para a variação do valor de ε ao longo do treino pretende-se assim criar as condições necessárias para que o Agente seja capaz de identificar repetidamente a configuração de parâmetros que permite à Meta-Heurística obter a melhor solução.

Relativamente aos restantes parâmetros que caracterizam o processo de treino - número de episódios e número de iterações por episódios, estes são adaptados para cada instância porque dependem da dimensão de cada uma, logo é crucial perceber a influência que a instância em si tem. Os limites dos intervalos de alguns parâmetros do SA dependem da dimensão da instância do problema. No exemplo da TI e do EL, estes são influenciados pelo espaço de soluções, e pelo do número de atividades e máquinas, respetivamente. Um estado é constituído pelos quatro parâmetros, o que significa que, em teoria, o número total de estados será maior quanto maior for a dimensão da instância. Como o objetivo é permitir que o Agente visite os diferentes estados e explore as várias ações em cada estado para perceber a influência que cada um tem na resolução do problema, quanto maior for o número de estados, maior é a necessidade de aumentar o número de iterações por episódio e o número de episódios para permitir ao Agente não só explorar como também consolidar a sua aprendizagem. Sendo assim, é crucial adaptar os valores destes parâmetros ao problema a resolver, o que será abordado no estudo computacional que constitui o Capítulo 4.

A Figura 25 apresenta o diagrama representativo do processo de treino do Agente, que tem como base todos os princípios mencionados ao longo deste Capítulo 3. O primeiro episódio é utilizado com o objetivo de definir um valor inicial representativo da melhor solução encontrada pelo conjunto "AP + MH". De seguida, de forma geral, a sequência inicia-se com a escolha de uma ação segundo a estratégia de seleção *epsilon-greedy*, que provoca uma alteração do estado atual para uma nova configuração de parâmetros. O problema *Job-Shop* é resolvido através da execução de 5 corridas do SA, e o valor da melhor solução (em termos de *makespan*) resulta da média desses valores. Em cada momento, verifica-se se é necessário atualizar o valor da melhor solução, que influencia o modo como a recompensa é calculado, e, posteriormente, procede-se à atualização dos valores da tabela Q com base nos princípios que caracterizam o algoritmo. Para concluir uma iteração de um episódio, é necessário perceber em que fase o Agente se encontra para atualizar o valor de *epsilon*, porque este diminui à medida que o treino progride – se na 1º Fase, onde ainda não foi alcançado o limite de episódios consecutivos sem melhoria da melhor solução, na 2º Fase, subdividida em quatro subfases no qual o *epsilon* diminui à medida que o treino avança, ou na 3º Fase, que corresponde aos últimos 40% dos episódios. De seguida, analisa-se se alguns dos critérios de interrupção definidos é verificado. O fim do treino verifica-se quando o número limite de episódios for atingido.

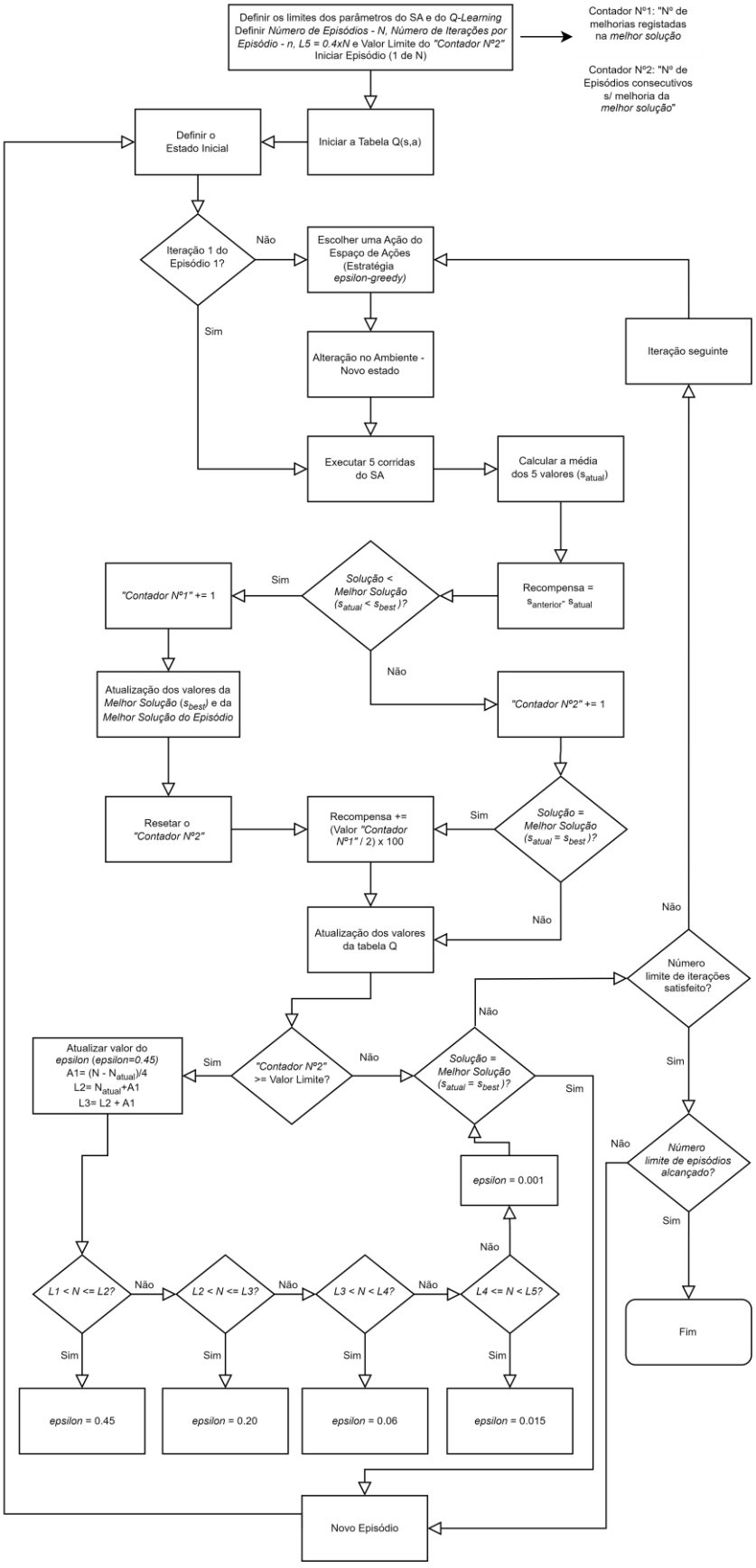


Figura 25- Diagrama representativo do processo de treino do Agente

Com os princípios que estão na base da formalização do Ambiente e das ações, com as diferentes estratégias de critério de interrupção, a função de recompensa e o modo como o processo de treino está modelado, pretende-se assim construir uma abordagem que seja capaz de parametrizar o SA. Numa fase inicial e/ou intermédia, como o conhecimento do Agente é nulo ou praticamente nulo, existe uma quantidade significativa de estados e ações para cada estado, logo é fundamental permitir que o Agente explore durante um período significativo de episódios e encontre a melhor solução. Posteriormente, o Agente também explora, mas o que mais se destaca é a sua capacidade de "utilizar" o seu conhecimento para perceber se, ao longo do tempo, é capaz de encontrar a configuração de parâmetros que conduz o SA à melhor solução, para que todo o sistema possa assim ser aplicado na resolução de Problemas de Otimização reais, no qual, para encontrar a melhor solução, apenas existe o conhecimento das características do problema.

4. Estudo Computacional, Resultados e Discussão

O Capítulo 4 apresenta o processo de validação do Protótipo do *Framework* de Parametrização de Meta-Heurísticas através de RL. É apresentado o estudo computacional para análise e validação dos resultados obtidos pelo Agente, incluído a apresentação das instâncias utilizadas do Problema de Minimização do *Makespan* em *Job-Shop*, e a comparação do desempenho do Protótipo com a informação presente na literatura, nomeadamente no que diz respeito à melhor solução, em termos de *makespan*. Para perceber o tipo de aprendizagem do Agente e validar assim a abordagem proposta neste trabalho, é analisada a evolução do número de iterações por episódios e a qualidade de cada combinação de parâmetros do SA.

4.1. Estudo Computacional

O Protótipo do *Framework* de Parametrização foi desenvolvido para resolver Problemas de Escalonamento. A máquina utilizada na implementação do Protótipo e na realização do estudo computacional é um HP Laptop, com um Processador Intel Core i7-1065G7 CPU @ 1.30GHz (8 CPUs) ~1.5GHz, 16.0 GB de RAM, Placa Gráfica NVIDIA GeForce MX330 e Sistema Operativo Microsoft Windows 11 Home de 64 bitz. Neste estudo computacional, para testar o desempenho do sistema, foram utilizadas instâncias representativas de Problemas de *Job-Shop*, onde o objetivo é a Minimização do tempo de conclusão, o *Makespan*, que é um dos objetivos mais comuns tanto nas aplicações reais como nos estudos desenvolvidos pelos autores na resolução deste tipo de problema [13]. Foram consideradas as instâncias disponíveis no *OR-Library*, que é uma coleção de dados de teste para uma variedade de problemas de Investigação Operacional que também disponibiliza as melhores soluções conhecidas para cada instância, o que permite assim avaliar a qualidade dos resultados obtidos neste estudo [7].

Considerando os objetivos estabelecidos para a abordagem apresentada neste trabalho, foram selecionadas instâncias de problemas apresentadas por Lawrence [285]. Estas instâncias caracterizam-se pelas iniciais do autor e pelo número da instância, sendo que quanto maior for esse número, maior é a dimensão do problema. Cada instância possui um determinado número de atividades para processar num número de máquinas, sendo cada uma constituída por um número de operações igual ao número de máquinas.

A escolha de instâncias com diferentes dimensões propostas por Lawrence [285] teve em consideração os princípios fundamentais do funcionamento do Protótipo do *Framework* de

Parametrização, particularmente no que se refere ao algoritmo implementado. Numa fase inicial do processo de treino, em que é essencial obter feedback rápido do comportamento de todo o sistema, é vantajoso utilizar instâncias de menor dimensão, podendo-se citar, como exemplo, 10x5 (10 atividades e 5 máquinas) e 15x5 (15 atividades e 5 máquinas). Após a realização dos ajustes necessários, torna-se necessário avaliar a eficiência do Protótipo na resolução de problemas *Job-Shop* de maior dimensão, na qual se pode incluir as instâncias 20x5 (20 atividades e 5 máquinas) e 10x10 (10 atividade se 10 máquinas).

Após a escolha das instâncias que serão utilizadas no estudo computacional, é possível concluir o processo de definição dos valores dos parâmetros que direta ou indiretamente dependem da dimensão das mesmas. A Tabela 11 apresenta os intervalos e/ou valores dos parâmetros TI e EL. Para ambos os parâmetros, os cálculos são realizados automaticamente pelo próprio Protótipo do *Framework* de Parametrização após a instância ser lida pelo Leitor de Instâncias. O EL apenas depende da dimensão da instância, portanto todas as instâncias com a mesma dimensão possuem o mesmo intervalo. Para o caso da TI, poder-se-ia aplicar o mesmo critério, no entanto, na execução das 5000 iterações para observar o espaço de soluções, verificou-se que, para instâncias com a mesma dimensão, existiam variações nos valores dos limites. Sendo assim, para evitar que isso influencie negativamente o processo de obtenção da melhor solução, definiu-se um intervalo específico para cada instância. O CR e a EV são definidos de forma universal para todas as instâncias, sendo $[0.80, 0.99]$ e $[\text{'}TRANSPOSE\text{'}, \text{'}SWAP\text{'}, \text{'}INSERT\text{'}, \text{'}REVERSE\text{'}]$, respetivamente.

Tabela 11- Intervalos dos parâmetros que caracterizam o SA nas instâncias la01 a la20

Instância	Dimensão	Temperatura Inicial	*Epoch Length*
la01	10 x 5	$[4800, 10100]$	$[47, 53]$
la02		$[4800, 10300]$	
la03		$[4400, 9500]$	
la04		$[4600, 9600]$	
la05		$[3900, 8300]$	
la06	15 x 5	$[6100, 13000]$	$[71, 79]$
la07		$[6300, 13300]$	
la08		$[6100, 13000]$	
la09		$[6400, 13600]$	
la10		$[6100, 12900]$	
la11	20 x 5	$[7800, 16600]$	$[95, 105]$
la12		$[6900, 14700]$	
la13		$[7600, 16100]$	
la14		$[7800, 16500]$	
la15		$[8500, 18000]$	
la16	10 x 10	$[7100, 15100]$	$[95, 105]$
la17		$[6300, 13300]$	
la18		$[6700, 14200]$	
la19		$[6900, 14500]$	
la20		$[7100, 15000]$	

A Tabela 12 apresenta os valores do Critério de Interrupção do SA, representado pelo número de iterações, e, relativamente ao processo de treino, do número de episódios, do número de iterações por episódio e do número de episódios sem melhoria do *makespan*. A definição dos valores de cada parâmetro compreendeu a realização de diversos testes. No caso do Critério de Interrupção do SA, o objetivo foi definir um valor que assegure uma convergência consistente, capaz de minimizar a variabilidade nos resultados em execuções repetidas do algoritmo. Caso a variabilidade seja significativa, o Agente pode obter valores completamente distintos para a mesma configuração de parâmetros, o que prejudica a obtenção de bons resultados. Para o número de episódios e o número de iterações por episódio, ambos os valores aumentam à medida que dimensão da instância aumenta. Sendo assim, à medida que a dimensão da instância aumenta, como existem mais estados, é fundamental aumentar o número de iterações para permitir ao Agente uma exploração eficaz, e, consequentemente, uma aprendizagem consolidada sobre quais são as melhores ações em cada estado. Como os limites dos intervalos dos restantes parâmetros são calculados de acordo com o recomendado na literatura, ou seja, de certa forma o espaço é restringido e pode ser considerado como um que possivelmente permite obter soluções boas, próximas da melhor em cada instância, o número de iterações não apresenta dimensões tão elevadas como à partida seria expectável por ser comum em outros problemas de RL.

Tabela 12- Valores dos parâmetros que caracterizam o *Q-Learning* e o processo de treino

Instância	**Critério de Interrupção SA (Iterações)**	**Nº de Episódios**	**Nº Iterações p/Episódio**	**Nº Episódios s/melhoria do *makespan***
la01	3500	250	300	20
la02	22500	200	200	20
la03	35000	125	225	20
la04	22500	200	200	20
la05	2750	250	300	20
la06	3000	275	350	25
la07	4500	275	325	25
la08	3500	275	325	25
la09	3500	275	325	25
la10	2750	275	325	25
la11	4000	300	400	30
la12	4000	300	400	30
la13	5000	300	400	30
la14	3750	300	400	30
la15	7500	300	400	30
la16	27500	125	225	20
la17	35000	125	225	20
la18	27500	125	225	20
la19	35000	125	225	20
la20	27500	125	225	20

Os valores do Critério de Interrupção do SA, representado pelo número de iterações executadas numa corrida da Meta-Heurística, sofreu um processo de experiências de calibração para encontrar um valor que permita testar a capacidade e utilidade do protótipo. O objetivo inicial era utilizar um valor igual para instâncias com a mesma dimensão, no entanto, os resultados iniciais evidenciaram que isso não seria possível porque, em algumas situações, os 5 resultados obtidos na execução da Meta-Heurística para uma determinada configuração resultavam em resultados significativamente diferentes, ou seja, caracterizados por uma variabilidade excessiva que influenciava negativamente o processo de aprendizagem do Agente.

A Figura 26 apresenta os resultados obtidos na resolução da instância la02 com a configuração de parâmetros – $(6800, 0.80, 52, 'SWAP')$, no qual foram executadas 3500 iterações, que é o valor utilizado para a instância la01 e que permitiu obter resultados excelentes. A análise dos resultados evidencia uma clara variação dos valores. Caso isto aconteça de forma recorrente durante o processo de treino do Agente, existe uma elevada possibilidade de ele encontrar a melhor solução com uma configuração de parâmetros, ser recompensado por isso, no entanto, quando voltar a este estado, pode-se obter resultados completamente diferentes. Sendo assim, instâncias com a mesma dimensão apresentam valores do Critério de Interrupção diferentes para permitir que a execução da Meta-Heurística não influencie negativamente o papel do Agente. Para cada instância aumentou-se gradualmente o número de iterações para perceber qual era o momento em que a variação era nula ou praticamente nula, evitando-se assim a utilização de valores demasiado elevados que permitam avaliar muitas soluções e encontrar as melhores soluções para diversas configurações de parâmetros, e valores excessivamente baixos que impeçam a exploração eficiente do espaço de soluções. O aumento do número iterações da Meta-Heurística tem como consequência um aumento do tempo de processamento. Sendo assim, para obter resultados num período exequível e, ao mesmo tempo, alcançar soluções de qualidade, nas instâncias la02 a la04 e la16 a la20, que apresentam valores significativamente elevados no Critério de Interrupção do SA, reduziu-se o número de episódios e o número de iterações por episódios para valores adequados que não prejudiquem o processo de aprendizagem do Agente.

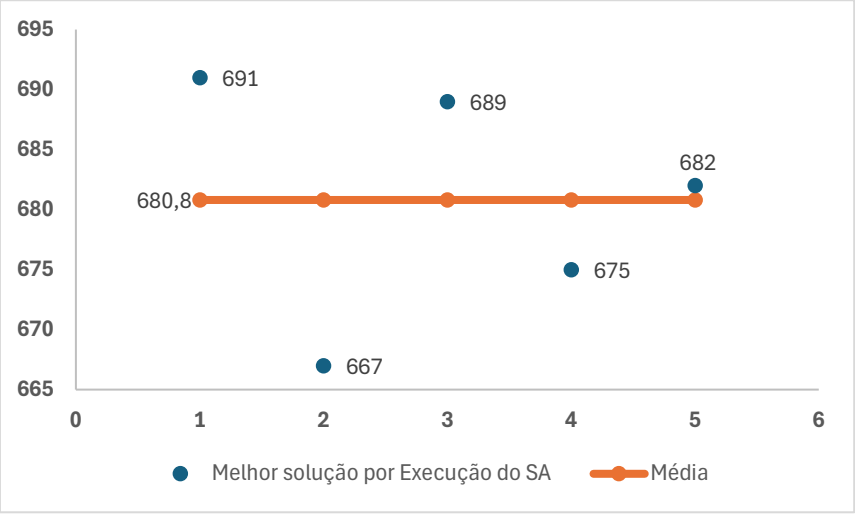


Figura 26- Resultados da execução da Meta-Heurística para a instância la02 com 3500 iterações como Critério de Interrupção do SA

O objetivo da abordagem desenvolvida neste trabalho é construir uma estrutura que permita a um Agente identificar a configuração de parâmetros que permita à Meta-Heurística obter a melhor solução na resolução de um Problema de Escalonamento. Definir uma estratégia que permita analisar detalhadamente e avaliar se o objetivo foi cumprido é um passo determinante. Em primeiro lugar, após a execução de todos os episódios, a melhor solução, em termos de *makespan*, obtida pelo conjunto "AP + MH", será comparada com a solução mais otimizada disponível na literatura. No entanto, para além de perceber se a melhor solução foi obtida, é fundamental avaliar se o Agente foi capaz de aprender uma política que, ao fim de um determinado número de episódios, lhe permite alcançar essa solução repetidamente. Em problemas que envolvem a utilização de RL para resolver um problema, é comum fazer esta avaliação através da análise da curva de aprendizagem do Agente, que retrata a evolução da recompensa acumulada de cada episódio [291]. Em exemplos como o "*GridWorld Problem*", onde o objetivo é que o Agente seja capaz de alcançar um "objetivo" no menor número de passos possíveis, a curva representativa da recompensa acumulada tende a aumentar, e, após um determinado número de iterações, estabiliza e observa-se uma convergência para esse valor. No caso da abordagem apresentada neste trabalho, a adição das componentes relacionadas com a Meta-Heurística e o funcionamento do processo de treino, fazem deste tipo de análise uma tarefa complexa. A fórmula de cálculo da recompensa incentiva o Agente a explorar combinações de parâmetros que conduzam a valores menores de *makespan*, e penaliza-o caso a sua ação o tenha conduzido a valores maiores, no entanto, devido à natureza estocástica das Meta-Heurísticas, é expectável a existência de oscilações nas recompensas em cada iteração, e, consequentemente, na recompensa acumulada de cada episódio. Por outras palavras, num determinado estado, devido à aleatoriedade existente no processo de geração e avaliação de soluções, a escolha da mesma ação pode provocar recompensas diferentes, o que influencia o valor da recompensa acumulada nesse episódio. Esta situação tem tendência a agravar-se quando os números de estados e de ações em cada estado são elevados porque pode acontecer ao longo de um número de iterações mais extenso. A abordagem proposta apresenta soluções para minimizar o efeito desta natureza estocástica, como a execução de várias corridas para a mesma configuração de parâmetros.

Relativamente ao processo de treino em si, o critério de interrupção de um episódio está modelado para criar um balanço entre o processo de exploração do Agente e a sua capacidade para encontrar a melhor solução. Numa fase inicial, o Agente não possui informação sobre nenhuma solução, encontra-se num processo de total descoberta, logo, como um episódio termina se ele for capaz de encontrar uma solução melhor, é expectável obter episódios mais curtos, com um número de iterações significativamente inferior quando comparado com os restantes. Quando o Agente encontra aquele que, mesmo ele não sabendo, será a melhor solução para aquela instância, e até atingir o valor limite definido, o único critério de interrupção é o número de iterações por episódio, logo esta fase caracteriza-se por episódios significativamente mais longos. De seguida, todos os episódios terminam se o Agente encontrar essa melhor solução, e, como é expectável que seja capaz de adquirir a capacidade para o encontrar rapidamente, o número de iterações será possivelmente baixo novamente, e, de certa forma, as recompensas acumuladas não apresentarão valores idênticos às dos episódios onde se realizam as iterações todas. O processo de recompensas é desenvolvido

especificamente para resolver o problema deste trabalho, e é preponderante para permitir que o Agente alcance os seus objetivos. No entanto, caracteriza-se por constantes oscilações que impactam diretamente a evolução da curva da recompensa acumulada. Sendo assim, apesar da curva de aprendizagem ser uma parte integrante importante do processo de análise, para avaliar a utilidade do Protótipo do *Framework* de Parametrização vão assim ser utilizadas as evoluções do melhor *makespan* obtido e do número de iterações, ambas em cada episódio.

4.2. Resultados do Protótipo do Framework de Parametrização em Problemas de Minimização do *Makespan* em *Job-Shop*

O critério de desempenho utilizado para resolver os Problemas *Job-Shop* é o *makespan*, que representa o tempo de conclusão da última operação numa determinada máquina de um plano de Escalonamento [13]. Desta forma, primeiro serão comparadas as soluções obtidas pelo Protótipo do *Framework* de Parametrização com as soluções ótimas disponíveis na literatura, que foram obtidas pelos diversos autores. Por fim, o processo de análise finaliza com as curvas de evolução da melhor solução (em termos de *makespan*) e do número de iterações por episódio, e da análise da qualidade das combinações de parâmetros que permitiram ao Agente obter a melhor solução, com o objetivo de avaliar a capacidade do Agente na resolução de Problemas de Escalonamento.

4.2.1. Qualidade das Soluções

As instâncias foram testadas no Protótipo do *Framework* de Parametrização por ordem crescente de dimensão. O desempenho do sistema foi medido pelo Desvio Percentual Relativo (RPD, do inglês *Relative Percentage Deviation*), calculado através da Equação (48). Neste caso, sol_{prot} representa a melhor solução encontrada pelo Protótipo implementado neste trabalho, enquanto $sol_{ótima}$ é o valor da melhor solução que está disponível em *OR-Library* [7].

$$RPD(\%) = \frac{sol_{prot} - sol_{ótima}}{sol_{ótima}} \times 100 \tag{48}$$

A Tabela 13 apresenta os resultados computacionais da resolução de 20 instâncias, no qual são apresentadas as melhores soluções obtidas, em termos de *makespan*, pelo Protótipo do *Framework* de Parametrização, e as soluções ótimas disponíveis em *ORLibrary* [7]. Na resolução das 20 instâncias, o sistema foi capaz de obter a melhor solução em 17, o que representa 85% de eficácia. Apenas em 3 das instâncias de maior dimensão (10x10) é que o Agente não obteve a melhor solução reportada na literatura, mas a diferença é significativamente baixa, com 1 unidade temporal na instância la16, 5 unidades temporais na instância la20 e 6 unidades temporais na instância la19.

Tabela 13- Comparação dos resultados obtidos pelo Protótipo com os valores da literatura nas instâncias la01 a la20

Instância	Dimensão	Melhor solução (Protótipo)	Solução Ótima (Literatura)	RPD (%)
la01	10 x 5	666	666	0.00
la02		655	655	0.00
la03		597	597	0.00
la04		590	590	0.00
la05		593	593	0.00
la06	15 x 5	926	926	0.00
la07		890	890	0.00
la08		863	863	0.00
la09		951	951	0.00
la10		958	958	0.00
la11	20 x 5	1222	1222	0.00
la12		1039	1039	0.00
la13		1150	1150	0.00
la14		1292	1292	0.00
la15		1207	1207	0.00
la16	10 x 10	946	945	0.11
la17		784	784	0.00
la18		848	848	0.00
la19		848	842	0.71
la20		907	902	0.55

4.2.2. Evolução do Melhor *Makespan* e do Número de Iterações por Episódio

O valor de *epsilon* influencia o modo como uma ação é escolhida - se aleatoriamente ou com base na tabela de valores Q. Numa fase inicial, o Agente não possui informação nem conhecimento sobre as soluções, logo necessita de explorar os diferentes estados e as diferentes ações em cada estado para realmente perceber o que o recompensará mais a longo prazo. Sendo assim, devido à quantidade significativa de estados e ações, nesta fase, o processo de treino do Agente caracteriza-se por valores de *epsilon* elevados. À medida que os valores deste parâmetro diminuem, o Agente continua a adquirir informação num processo caracterizado por exploração e aproveitamento. Posteriormente, quando os valores de *epsilon* são relativamente baixos e/ou iguais ao valor mínimo definido, significa que a probabilidade de, em cada estado, se recorrerem às ações com maior valor Q, aumenta significativamente, logo o Agente pode colocar em prática o conhecimento adquirido ao longo do processo de treino.

Para perceber se o objetivo estabelecido para o sistema é cumprido, são monitorizadas, para cada instância, a evolução da melhor solução (melhor *makespan*) e do número de iterações executadas em cada episódio. O primeiro caso mencionado tem um propósito evidente, que é perceber se o Agente foi capaz de obter a melhor solução. Por sua vez, o segundo aspeto é introduzido para analisar se o Agente realmente foi capaz de aprender uma política que lhe

permita obter esse valor repetidamente e, de preferência, num número de iterações cada vez menor à medida que os episódios avançam e o valor de *epsilon* diminui.

Tendo em consideração o facto de terem sido resolvidas diversas instâncias, de seguida, serão apresentados os resultados para as instâncias la01, la04, la06, la12 e la18 e la20. Com estas instâncias pretende-se apresentar os resultados para todos os tipos de dimensões analisados e, simultaneamente, abordar e justificar os diferentes resultados obtidos, nomeadamente em instâncias onde a dificuldade de resolução foi mais significativa. Os gráficos obtidos para as restantes instâncias são apresentados no Apêndice C.

Instância la01

A Figura 27 ilustra a evolução da melhor solução (melhor *makespan*) encontrada por episódio para a instância la01, que é uma das instâncias mais simples que representam problemas *Job-Shop*. O sistema é capaz de obter o melhor valor – 666, numa fase inicial do processo, o que é expectável devido à simplicidade da mesma, e ao facto dos limites dos intervalos de cada parâmetro, bem como o estado inicial, também conduzirem a um espaço que, à priori, leva a soluções próximas das melhores.

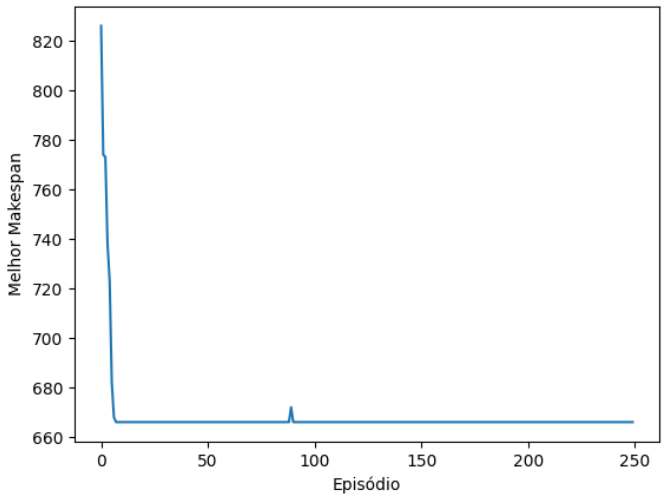


Figura 27- Evolução da melhor solução por episódio: instância la01

A Figura 28 ilustra a evolução da melhor solução (melhor *makespan*) do número de iterações e do valor de *epsilon* por episódio para a instância la01. Numa fase inicial, enquanto o valor de *epsilon* ainda é significativo – aproximadamente até ao episódio 100, o Agente encontra a melhor solução na maioria dos episódios, no entanto, ainda se verifica alguma oscilação no número de iterações, o que é justificável, em parte, porque muitas ações são aleatórias e o Agente encontra-se em processo de exploração. À medida que os episódios avançam, verificam-se melhorias significativas, o Agente já "assumiu" o valor da melhor solução como sendo a melhor, e apresenta a capacidade de o encontrar sucessivamente num número de iterações que decresce. A partir do momento que o *epsilon* apresenta o valor mínimo, o que corresponde aos últimos 100 episódios, o sistema consegue encontrar o melhor valor em todos os episódios num número de iterações significativamente baixo, praticamente sempre inferior a 30 iterações e, na maioria, abaixo das 20, o que comprova a sua capacidade de aprendizagem, bem como o facto de todo o processo de exploração ter sido bom o suficiente para lhe permitir alcançar isso.

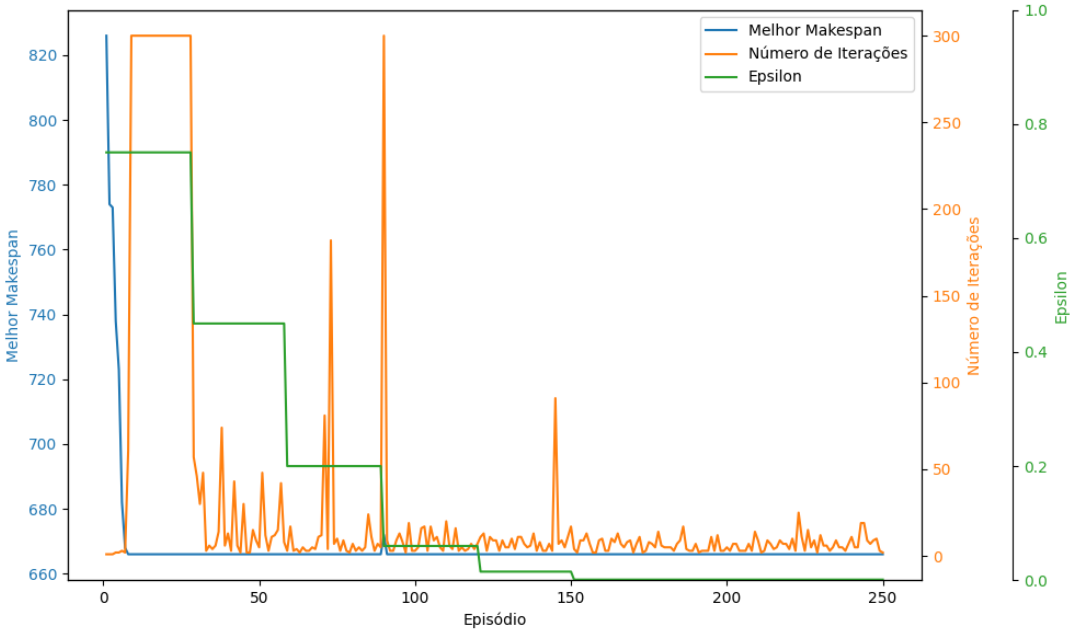


Figura 28- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la01

Instância la04

A Figura 29 apresenta a evolução da melhor solução (melhor *makespan*) encontrada para a instância la04. O Agente consegue alcançar o valor que se confirma como sendo a melhor solução numa fase inicial do treino - 590, e é capaz de o fazer sucessivamente, o que é expectável devido ao número de iterações utilizada como Critério de Interrupção do SA, que é significativamente elevado devido à irregularidade do espaço de soluções.

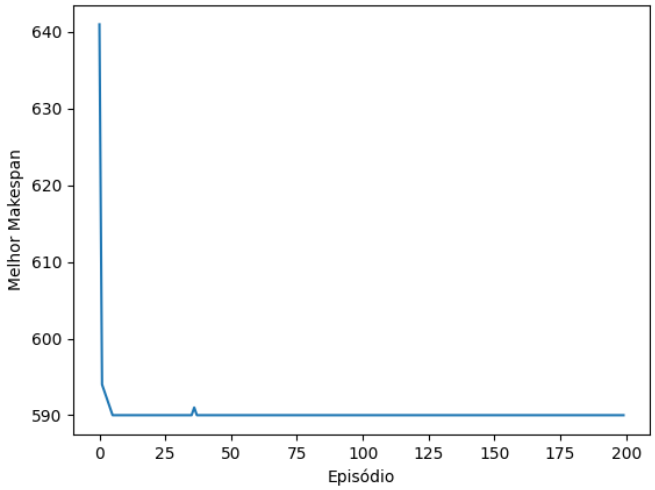


Figura 29- Evolução da melhor solução por episódio: instância la04

A Figura 30 ilustra a evolução da melhor solução (melhor *makespan*), do número de iterações e do valor de *epsilon* por episódio para a instância la04. À medida que o treino progride, nota-se que o Agente consegue, de facto, encontrar combinações de parâmetros que conduzam a Meta-Heurística à melhor solução, e o número de iterações necessário para o fazer apresenta

uma tendência clara de diminuição. Quando comparando o resultado obtido com os de outras instâncias, a partir do episódio 120, momento onde o *epsilon* possui o valor mínimo, verifica-se que, neste caso, o número de iterações ainda apresenta uma certa oscilação. A análise dos estados visitados pelo Agente e a execução da Meta-Heurística, de forma isolada, com cada configuração de parâmetros que o Agente prioriza, permite perceber que esta observação pode ser diretamente relacionada ao valor do número de iterações do Critério de Interrupção do SA. Embora o Critério de Interrupção definido tenha permitido ao Agente identificar e visitar, nesta fase, boas combinações de parâmetros, com um valor significativo que lhe permitiram obter a melhor solução, a natureza complexa do problema, nomeadamente o irregularidade do espaço de soluções, sugere que um Critério de Interrupção superior seria mais adequado para assegurar uma exploração mais profunda do espaço de soluções. Para contornar o facto da mesma configuração ainda não produzir consistentemente os mesmos resultados, o Agente adota ações de *"keep"* para manter-se nesse estado, ou procura alternativas que o reconduzam à mesma configuração ou a outra semelhante, o que tem como consequência um número um pouco menos constante e mais elevado de iterações. Mesmo assim, de forma geral, é evidente a capacidade de aprendizagem do Agente e a qualidade dos resultados alcançados.

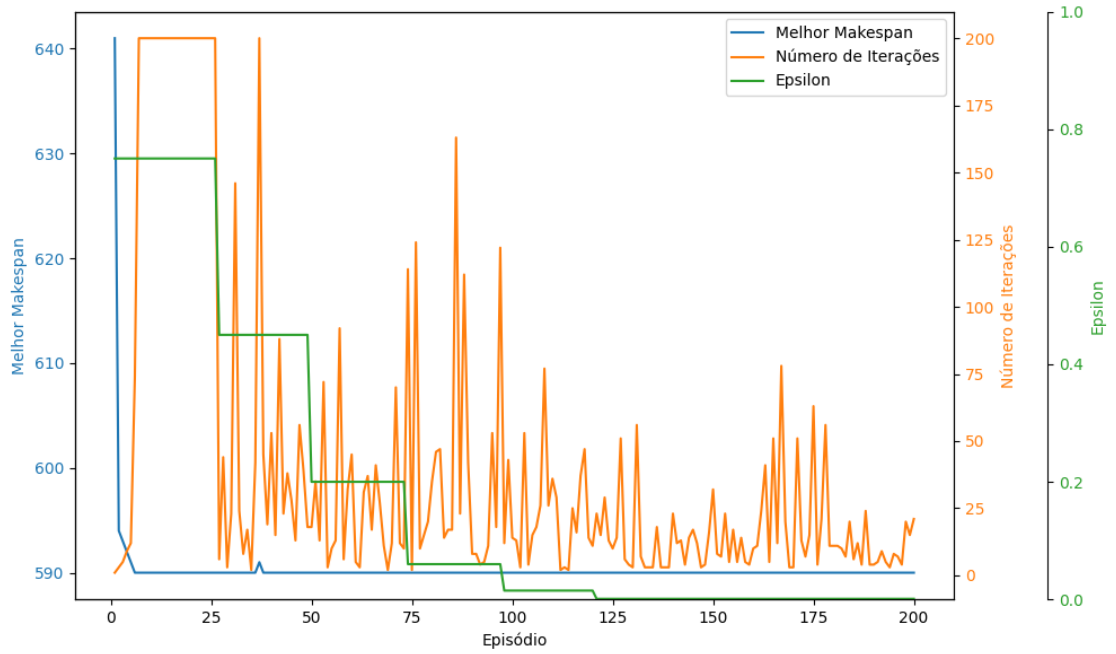


Figura 30- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la04

Instância la06

A Figura 31 apresenta a evolução da melhor solução (melhor *makespan*) encontrada para a instância la06. O Agente consegue alcançar o valor que se confirma como sendo o melhor *makespan* numa fase inicial do treino, durante os primeiros 20 episódios, no entanto, durante um período longo até ao episódio 158, a melhor solução por episódio oscila significativamente. A partir deste momento, o valor 926 assume-se claramente como sendo o valor que o Agente almeja alcançar e é capaz de o fazer sucessivamente ao longo dos episódios até ao fim do processo de pesquisa.

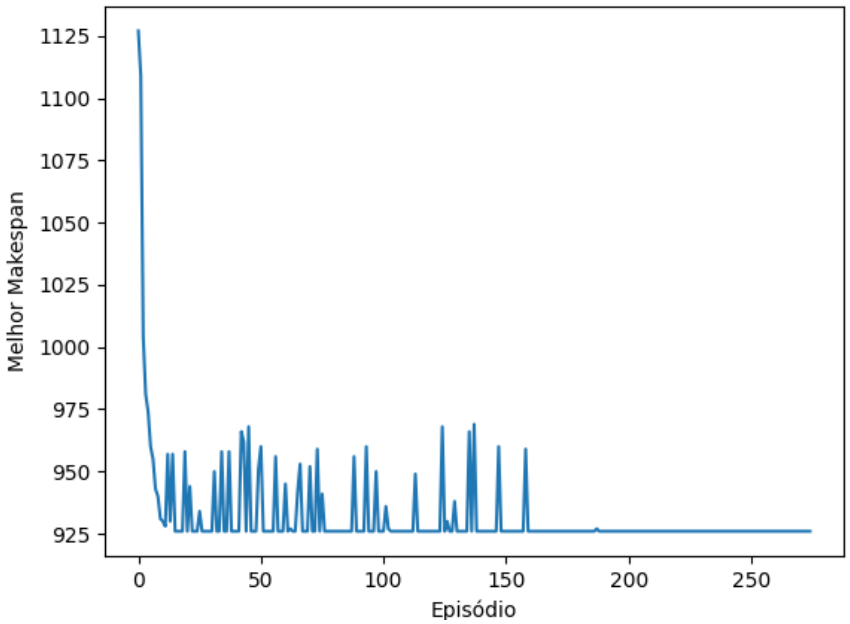


Figura 31- Evolução da melhor solução por episódio: instância la06

A Figura 32 ilustra a evolução da melhor solução (melhor *makespan*), do número de iterações e do valor de *epsilon* por episódio para a instância la06. Apesar do Agente encontrar a melhor solução numa fase inicial do processo de pesquisa, o número de iterações necessário para o fazer oscila significativamente à medida que o *epsilon* decresce. Quando o *epsilon* atinge o valor mínimo, o número de iterações diminui, mas ainda apresenta alguma variação durante aproximadamente 30 episódios, o que pode indicar que as combinações de parâmetros que o Agente procura não permitem que a Meta-Heurística obtenha a melhor solução sucessivamente para o valor do Critério de Interrupção definido. No entanto, a partir deste momento, este processo estabiliza e, ao longo dos restantes episódios, cerca de 85, a capacidade do Agente encontrar a melhor solução num número de iterações reduzido, inferior a 20 iterações, intensifica-se, o que evidencia de forma clara a sua capacidade de aprendizagem, uma vez que identificou uma combinação de parâmetros de qualidade e sabe como alcançá-la. Devido à natureza estocástica das Meta-Heurísticas, existem pequenas exceções no qual o Agente não é capaz de encontrar esse valor, no entanto é apenas um episódio, o que significa que, tendo em consideração o número total de episódios, é estatisticamente insignificante.

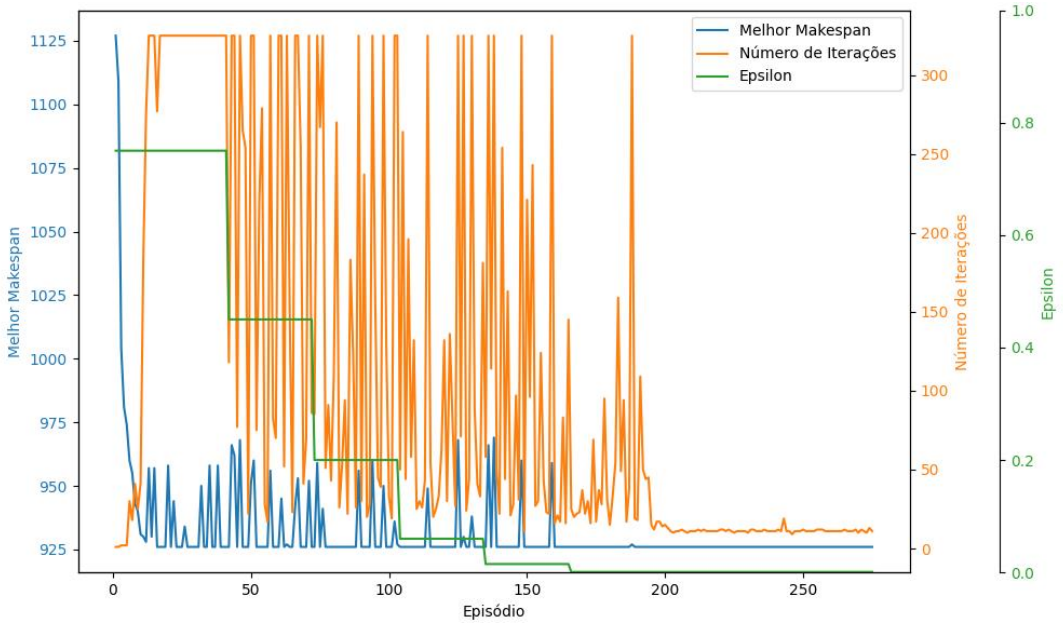


Figura 32- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la06

Instância la12

A Figura 33 apresenta a evolução da melhor solução (melhor *makespan*) encontrada para a instância la12. Apesar do Agente ser capaz de encontrar muito cedo o valor que se confirma como sendo o melhor *makespan*, ainda se verifica uma oscilação evidente até ao episódio 80 sensivelmente. A partir deste momento, o valor 1039 é, de forma evidente, o valor que o Agente identifica como o melhor e é capaz de o encontrar constantemente, com exceção de apenas 1 episódio que acontece numa fase intermédia do processo.

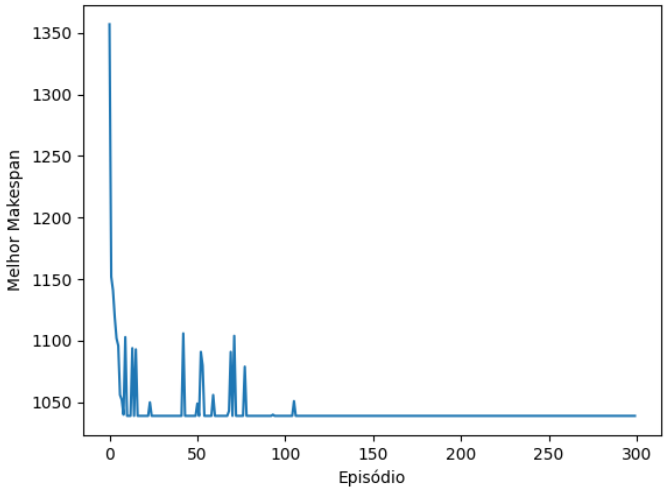


Figura 33- Evolução da melhor solução por episódio: instância la12

A Figura 34 apresenta a evolução do melhor *makespan*, do número de iterações e do valor de *epsilon* por episódio para a instância la12. Para valores elevados de *epsilon*, o processo de aquisição de informação é caracterizado pela execução de um número significativo de iterações. Quando esse parâmetro apresenta valores próximos a 0.1, esta tendência inverte-se

totalmente, com o Agente a ser capaz de encontrar aquele que é definido por ele como sendo a melhor solução, num número de iterações reduzido, praticamente sempre inferior a 20 iterações. Tendo em conta a dimensão da instância, o facto deste número ser significativamente baixo evidencia que o processo de treino do Agente foi capaz de lhe fornecer a informação necessária para ele alcançar os seus objetivos.

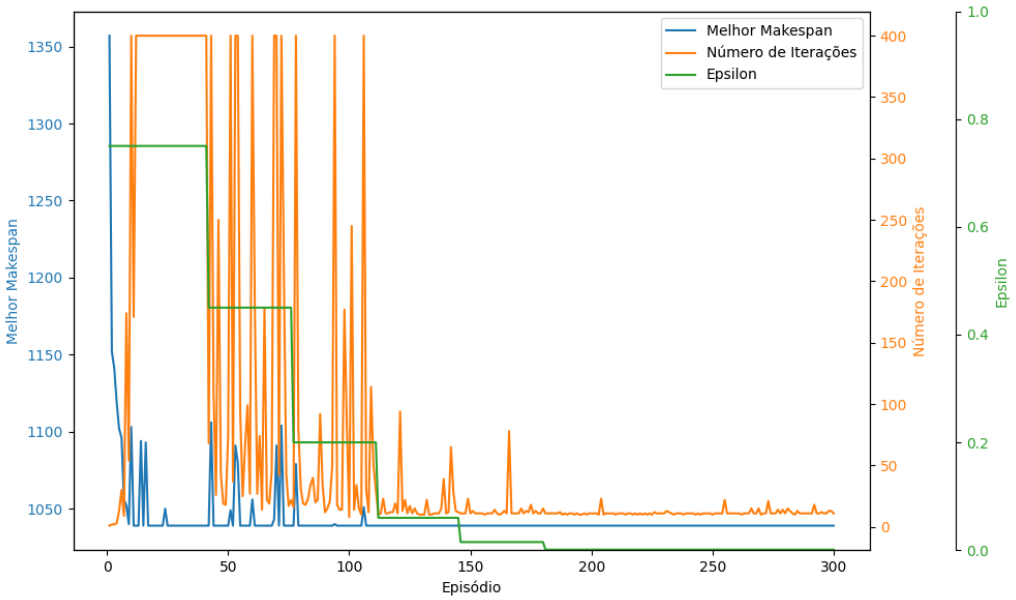


Figura 34- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la12

Instância la18

A Figura 35 apresenta a evolução da melhor solução (melhor *makespan*) encontrada para a instância la18. Observa-se que, embora a melhor solução identificada pelo Agente seja igual à melhor solução reportada até ao momento - 848, existe uma clara dificuldade em encontrá-la consistentemente ao longo de todo o processo de pesquisa, o que se traduz em resultados um pouco inconsistentes.

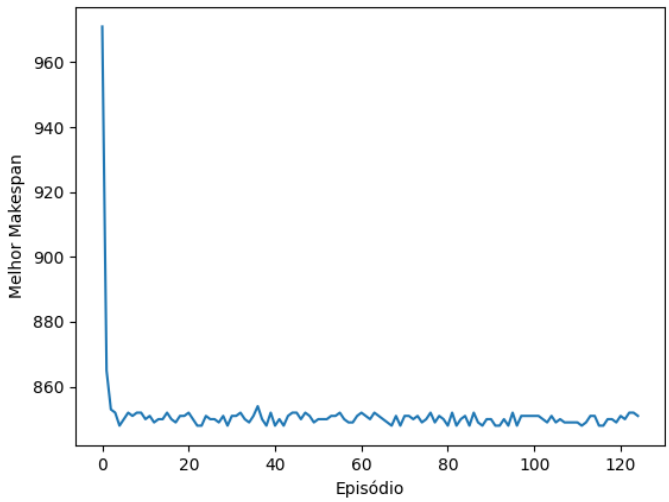


Figura 35- Evolução da melhor solução por episódio: instância la18

A Figura 36 apresenta a evolução do melhor *makespan*, do número de iterações e do valor de *epsilon* por episódio para a instância la18. Para que o Agente consiga adquirir conhecimento que lhe permita obter a melhor solução consistentemente quando o valor de *epsilon* é mínimo, é crucial que consiga explorar o suficiente e encontre a melhor solução em diferentes fases do processo. Para a instância la18, os resultados evidenciam uma clara dificuldade do Agente em encontrar uma configuração de parâmetros que produza resultados consistentes, uma vez que se observar uma grande variabilidade no número de iterações por episódio. As análises dos estados visitados pelo Agente e da qualidade de cada configuração identificada revelam que, tanto a variação do valor da melhor solução como a oscilação no número de iterações, não parecem estar diretamente correlacionados com os parâmetros que caracterizam o processo de treino, mas sim com a natureza dos parâmetros que caracterizam a Meta-Heurística. Especificamente, o valor do Critério de Interrupção utilizado não é suficiente para permitir que o SA explore de maneira abrangente o espaço de soluções porque o espaço de soluções pode ser particularmente complexo ou irregular para esta instância, o que faz com que o SA dependa de um Critério de Interrupção muito elevado. Assim, mesmo quando o Agente identifica uma boa configuração de parâmetros, esta é boa em determinados momentos, mas não é robusta o suficiente para garantir a estabilidade e a qualidade da solução ao longo do tempo.

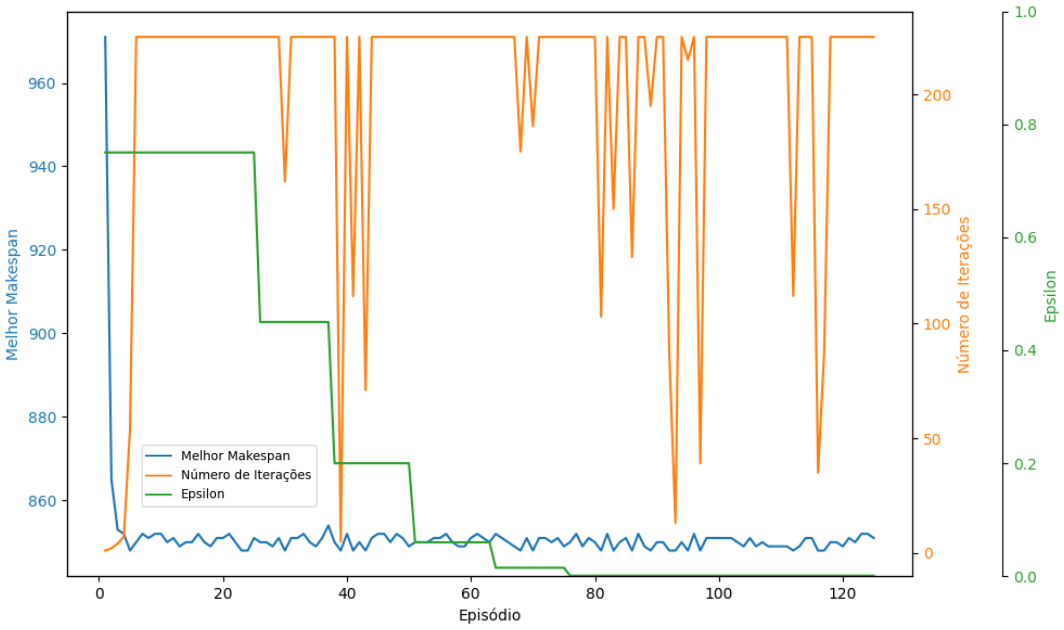


Figura 36- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la18

Instância la20

A Figura 37 apresenta a evolução da melhor solução (melhor *makespan*) encontrada para a instância la20. A resolução desta instância é representativa de um caso no qual a melhor solução encontrada pelo Agente – 907, embora seja um bom resultado, é ligeiramente superior ao melhor valor reportado na literatura, que é 902. Mesmo assim, apesar dessa solução não ser igual à melhor, é notório que o Agente consegue obter o seu melhor valor identificado ao longo de praticamente todo o processo de pesquisa.

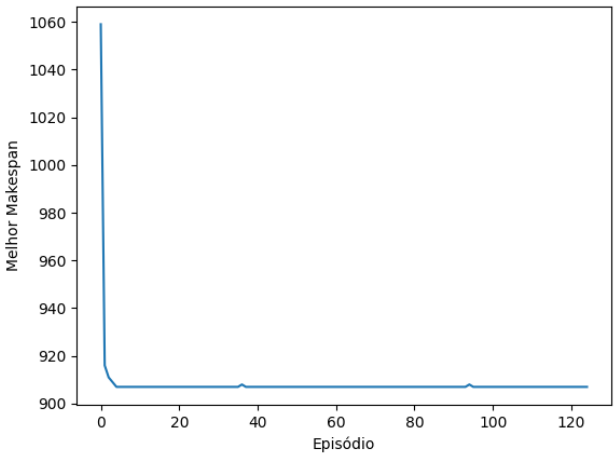


Figura 37- Evolução da melhor solução por episódio: instância la20

A Figura 38 apresenta a evolução do melhor *makespan*, do número de iterações e do valor de *epsilon* por episódio para a instância la20. A partir do momento em que o Agente identifica uma solução como "a melhor para a instância", ele consegue encontrá-la em praticamente em todos os episódios subsequentes, no entanto, o número de iterações oscila significativamente ao longo de todo o período analisado, independentemente da fase em que o Agente se encontra, ou seja, do valor do *epsilon*. Esta variação sugere que, em muitos casos, o Agente não conseguiu convergir para a melhor solução de maneira eficiente, possivelmente devido ao tempo insuficiente para a exploração completa do espaço de soluções por parte do SA. Em Meta-Heurísticas como o SA, o Critério de Interrupção é um parâmetro crítico que impacta diretamente a qualidade das soluções encontradas. Embora, para esta instância, o valor definido seja considerável, a análise dos estados visitados e das combinações de parâmetros priorizadas, permite perceber que ele não é suficientemente elevado para garantir que a mesma configuração de parâmetros produza consistentemente os mesmos resultados.

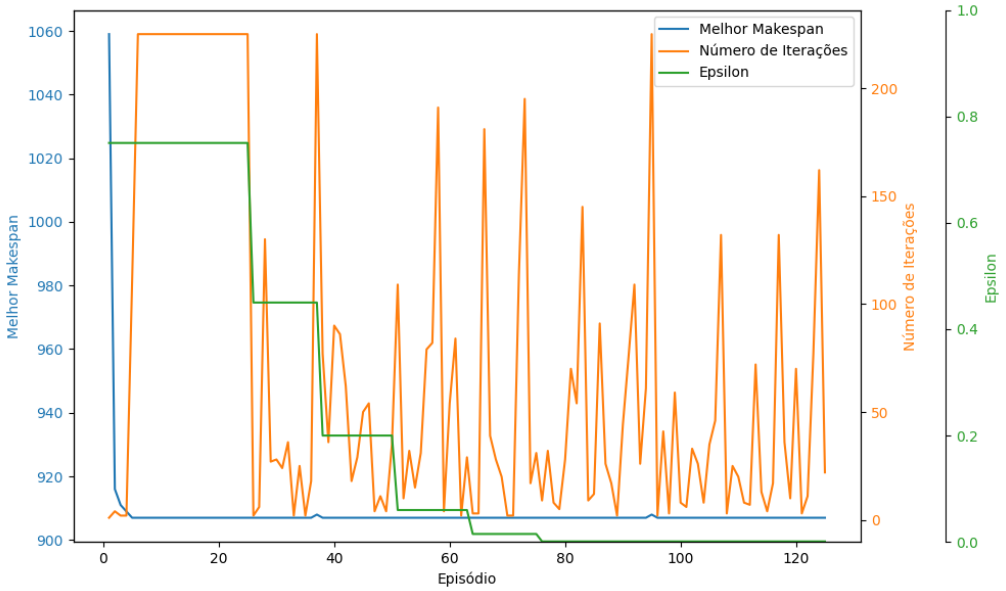


Figura 38- Evolução da melhor solução (*makespan*), número de iterações e valor de *epsilon* por episódio: instância la20

4.2.3. Qualidade da Configuração Ótima de Parâmetros

Identificar a melhor solução para uma determinada instância de um problema é um dos principais objetivos da abordagem proposta neste trabalho, no entanto, para além disso, também é crucial perceber a qualidade da configuração de parâmetro que a permitiu obter.

Durante o processo de resolução da instância, foram registadas as combinações de parâmetros que permitiram ao Agente obter a melhor solução a partir do momento que o valor de *epsilon* atingiu o valor mínimo, o que corresponde aos últimos 40% dos episódios em cada instância. Estes valores foram analisados para perceber se o Agente consegue, para cada instância, optar por uma configuração que lhe permita obter a melhor sucessão sucessivamente. Nas situações onde isso não foi possível e o Agente identificou várias configurações de parâmetros, estas foram analisadas para identificar as pequenas variações que existem.

A Tabela 14 apresenta as combinações de parâmetros que permitiram obter as melhores soluções para cada instância, bem como o número de episódios analisados e as respetivas frequências absoluta e relativa. Para cada instância, apenas são apresentadas as combinações de parâmetros que apresentam uma frequência significativa para o número de episódios analisados. Por exemplo, na instância la13, as duas combinações identificadas têm frequências de 74 e 34, respetivamente, representam 108 episódios, ou seja, 90% do número total de episódios (120). Nos restantes 12 episódios, o Agente alcançou a melhor solução através de duas combinações de parâmetros diferentes, mas que apresentam frequências de 8 e 4, o que não é significativo. Em termos de resultados obtidos, para cada instância, as combinações de parâmetros com frequências significativas apresentam pouca variação entre si. Essas variações, embora presentes, são mínimas e refletem ajustes relativamente pequenos para cada parâmetro, correspondendo à variação provocada por uma ação de "UP" ou "DOWN". A instância la16 é um caso particular, no qual o Agente não conseguiu obter a melhor solução nos últimos 50 episódios, que é o período em análise para esta instância.

Tabela 14- Frequências das combinações de parâmetros que conduzem à melhor solução nas instâncias la01 a la20

Instância	Parâmetros				Nº Episódios Analisados	Frequência Absoluta	Frequência Relativa (%)
	TI	CR	EL	EV			
la01	7300	0.89	51	*INSERT*	100	73	73.00
	7200	0.90	51	*INSERT*		26	26.00
la02	7600	0.87	51	*INSERT*	80	9	11.25
	7400	0.87	52	*INSERT*		9	11.25
	7400	0.86	52	*SWAP*		9	11.25
	7500	0.86	51	*SWAP*		8	10.00
	7400	0.89	51	*SWAP*		8	10.00
la03	7100	0.89	50	SWAP	50	12	24.00
	7000	0.89	50	SWAP		11	22.00

Instância	Parâmetros				Nº Episódios Analisados	Frequência Absoluta	Frequência Relativa (%)
	TI	CR	EL	EV			
la04	7000	0.86	48	*SWAP*	80	18	22.50
	6900	0.86	47	*SWAP*		14	17.50
	7000	0.90	50	*SWAP*		11	13.75
	7100	0.89	49	*SWAP*		9	11.25
la05	5900	0.90	50	*INSERT*	100	92	92.00
	5900	0.89	51	*INSERT*		8	8.00
la06	9300	0.83	76	*SWAP*	110	48	43.64
	9200	0.82	75	*SWAP*		17	15.45
la07	9500	0.85	74	*INSERT*	110	62	56.36
	9400	0.85	73	*INSERT*		44	40.00
la08	9600	0.82	78	*INSERT*	110	37	33.64
	9500	0.83	77	*INSERT*		31	28.18
	9700	0.83	76	*INSERT*		10	9.09
la09	9900	0.85	75	*INSERT*	110	31	28.18
	9800	0.86	75	*SWAP*		27	24.55
	9900	0.86	75	*INSERT*		25	22.73
la10	9500	0.83	71	*SWAP*	110	39	35.45
	9500	0.85	74	*SWAP*		27	24.55
	9500	0.85	73	*SWAP*		24	21.82
la11	12800	0.82	101	*INSERT*	120	41	34.17
la12	10500	0.83	97	*SWAP*	120	79	65.83
la13	11800	0.86	98	*SWAP*	120	74	61.67
	11900	0.86	99	*INSERT*		34	28.33
la14	11500	0.85	99	*INSERT*	120	37	30.83
	11400	0.85	98	*SWAP*		36	30.00
	11500	0.86	98	*INSERT*		28	23.33
la15	12900	0.85	101	*INSERT*	120	42	35.00
	13000	0.85	103	*INSERT*		38	31.67
la16	-	-	-	-	50	-	-
la17	9700	0.90	99	*SWAP*	50	4	8.00
	9700	0.80	103	*SWAP*		1	2.00
	9600	0.92	97	*SWAP*		1	2.00
	9500	0.90	103	*SWAP*		1	2.00

Instância	Parâmetros				Nº Episódios Analisados	Frequência Absoluta	Frequência Relativa (%)
	TI	CR	EL	EV			
la17	9500	0.82	98	*SWAP*	50	1	2.00
	9300	0.82	101	*SWAP*		1	2.00
	9200	0.95	99	*SWAP*		1	2.00
	8500	0.88	102	*INSERT*		1	2.00
	10000	0.92	100	*SWAP*		1	2.00
	10000	0.93	103	*INSERT*		1	2.00
	10400	0.96	102	*INSERT*		1	2.00
	10400	0.95	100	*SWAP*		1	2.00
	11200	0.90	101	*SWAP*		1	2.00
la18	10600	0.92	100	*INSERT*	50	1	2.00
	10500	0.87	98	*INSERT*		1	2.00
	10500	0.88	98	*INSERT*		1	2.00
	10500	0.95	98	*INSERT*		1	2.00
	10400	0.89	101	*INSERT*		1	2.00
	10400	0.82	105	*INSERT*		1	2.00
	10300	0.92	100	*INSERT*		1	2.00
	10300	0.91	97	*SWAP*		1	2.00
	10100	0.92	97	*SWAP*		1	2.00
	10100	0.80	104	*INSERT*		1	2.00
	9700	0.85	103	*INSERT*		1	2.00
la19	12000	0.93	102	*INSERT*	50	1	2.00
	11200	0.87	103	*INSERT*		1	2.00
	11000	0.95	97	*INSERT*		1	2.00
	10900	0.86	100	*SWAP*		1	2.00
	10800	0.89	104	*SWAP*		1	2.00
	10700	0.91	101	*INSERT*		1	2.00
	10600	0.88	105	*SWAP*		1	2.00
	10500	0.93	99	*SWAP*		1	2.00
	10500	0.85	102	*SWAP*		1	2.00
	9800	0.80	105	*INSERT*		1	2.00
la20	10900	0.86	101	SWAP	50	4	8.00
	10900	0.88	101	SWAP		3	3.75
	10900	0.89	101	SWAP		2	2.50
	10800	0.88	100	INSERT		2	2.50

O processo de treino modelado para o Protótipo do *Framework* de Parametrização permite que o Agente explore o suficiente o espaço de soluções e identifique algumas configurações de parâmetros que, de forma consistente, conduzem a Meta-Heurística à melhor solução. Para obter uma única configuração de parâmetros dominante, seria necessário utilizar valores significativamente elevados para o Critério de Interrupção da Meta-Heurística, no entanto, essa abordagem aumentaria o tempo de execução, e, de certa forma, comprometeria o interessa na análise da capacidade de aprendizagem do Agente, já que ele seria capaz de encontrar a melhor solução com facilidade. Além disso, é importante mencionar que, em instâncias com uma complexidade superior de resolução, especialmente aquelas com um espaço de soluções mais irregular, o Agente tende a utilizar um número maior de combinações de parâmetros. As instâncias la02 e la04 são um caso particular desta observação, no qual existem 5 e 4 combinações de parâmetros, respetivamente, com frequências semelhantes. No caso das instâncias de maior dimensão – la17 a la20, não apenas se verifica várias combinações, como as suas frequências também são muito baixas. Por exemplo, na instância la18, todas as configurações apenas permitiram à Meta-Heurística obter a melhor solução em uma única ocasião. Apesar do Critério de Interrupção já ser um valor significativamente elevado, ainda é difícil e desafiador encontrar uma configuração de parâmetros de alta qualidade que permita à Meta-Heurística obter consistentemente a melhor solução, o que evidencia assim a dificuldade adicional enfrentada pelo Agente em ambientes de alta complexidade, onde a diversidade de soluções viáveis é maior.

Com o objetivo de perceber a qualidade de cada configuração, a Meta-Heurística foi executada 20 vezes, de forma isolada, com cada configuração de parâmetros, e registaram-se os valores máximos e mínimos, bem como a média das 10 melhores soluções, que são apresentados na Tabela 15. É importante mencionar que, devido à questão da aleatoriedade na geração de soluções que é comum em Meta-Heurísticas, os resultados apresentados para cada combinação podem diferir ligeiramente quando testados noutra ocasião, mas a variação não será significativa porque já é utilizada uma amostra considerável (20 execuções). De forma geral, nas instâncias onde existem combinações com frequências mais significativas, o que correspondem àquelas onde, durante a resolução, o Agente não enfrentou dificuldades significativas – la01, la05 a la15, a qualidade é bem significativa, com as médias das 10 melhores soluções a evidenciarem que as configurações identificadas são capazes de permitir a obtenção da melhor solução reportada na literatura para essa determinada instância. Nas instâncias la02, la04, la17 e la18, apesar da melhor solução das 20 execuções ser igual à solução ótima, a média difere ligeiramente, mas não é algo significativo. Por sua vez, nas restantes instâncias – la19 e la20, onde, durante o processo de pesquisa, o Agente não foi capaz de obter o melhor valor reportado na literatura, as 20 execuções com cada configuração evidenciam que é possível obter esse valor – como se verifica com 3 configuração de parâmetros na instância la19, no entanto, a média das 10 melhores é superior. Para estes dois últimos casos mencionados, verifica-se que, de forma geral, para cada instância, as combinações de parâmetros apresentam frequências mais similares, não existe uma que claramente se destaque, o que desde logo é indicativo desta pequena variabilidade. Se o número de iterações, representativo do Critério de Interrupção do SA, for aumentado, é expectável que a variabilidade reduza consideravelmente.

Tabela 15- Resultados da execução da MH com cada configuração ótima de parâmetros nas instâncias la01 a la20

Instância	Parâmetros					Soluções (20 execuções)			Melhor Solução Protótipo	Solução Ótima (Literatura)
	TI	CR	EL	EV	Freq.	Min	Máx	Média (10 melhores)		
la01	7300	0.89	51	*INSERT*	73	666	709	666	666	666
	7200	0.90	51	*INSERT*	26	666	728	670		
la02	7600	0.87	51	*INSERT*	9	655	671	657	655	655
	7400	0.87	52	*INSERT*	9	655	683	662		
	7400	0.86	52	*SWAP*	9	655	690	660		
	7500	0.86	51	*SWAP*	8	655	686	660		
	7400	0.89	51	*SWAP*	8	655	680	659		
la03	7100	0.89	50	SWAP	12	597	620	603	597	597
	7000	0.89	50	SWAP	11	597	629	603		
la04	7000	0.86	48	*SWAP*	18	590	613	592	590	590
	6900	0.86	47	*SWAP*	14	590	611	593		
	7000	0.90	50	*SWAP*	11	590	607	591		
	7100	0.89	49	*SWAP*	9	590	613	590		
la05	5900	0.90	50	*INSERT*	92	593	593	593	593	593
	5900	0.89	51	*INSERT*	8	593	593	593		
la06	9300	0.83	76	*SWAP*	47	926	941	926	926	926
	9200	0.82	75	*SWAP*	17	926	948	926		
la07	9500	0.85	74	*INSERT*	62	890	898	890	890	890
	9400	0.85	73	*INSERT*	44	890	919	890		
la08	9600	0.82	78	*INSERT*	37	863	911	863	863	863
	9500	0.83	77	*INSERT*	31	863	893	863		
	9700	0.83	76	*INSERT*	10	863	921	864		
la09	9900	0.85	75	*INSERT*	31	951	988	951	951	951
	9800	0.86	75	*SWAP*	27	951	992	951		
	9900	0.86	75	*INSERT*	25	951	985	952		
la10	9500	0.83	71	*SWAP*	39	958	958	958	958	958
	9500	0.85	74	*SWAP*	27	958	998	958		
	9500	0.85	73	*SWAP*	24	958	995	958		
la11	12800	0.82	101	*INSERT*	41	1222	1254	1222	1222	1222
la12	10500	0.83	97	*SWAP*	79	1039	1064	1039	1039	1039

Instância	Parâmetros					Soluções (20 execuções)			Melhor Solução Protótipo	Solução Ótima (Literatura)
	TI	CR	EL	EV	Freq.	Min	Máx	Média (10 melhores)		
la13	11800	0.86	98	*SWAP*	74	1150	1190	1150	1150	1150
	11900	0.86	99	*INSERT*	34	1150	1184	1150		
la14	11500	0.85	99	*INSERT*	37	1292	1314	1292	1292	1292
	11400	0.85	98	*SWAP*	36	1292	1313	1292		
	11500	0.86	98	*INSERT*	28	1292	1320	1292		
la15	12900	0.85	101	*INSERT*	42	1207	1244	1207	1207	1207
	13000	0.85	103	*INSERT*	38	1207	1245	1207		
la16	-	-	-	-	-	-	-	-	946	945
la17	9700	0.90	99	*SWAP*	4	784	835	788	784	784
	9700	0.80	103	*SWAP*	1	784	811	788		
	9600	0.92	97	*SWAP*	1	784	811	789		
	9500	0.90	103	*SWAP*	1	784	803	786		
	9500	0.82	98	*SWAP*	1	787	825	789		
	9300	0.82	101	*SWAP*	1	784	809	789		
	9200	0.95	99	*SWAP*	1	784	797	789		
	8500	0.88	102	*INSERT*	1	784	842	792		
	10000	0.92	100	*SWAP*	1	784	834	789		
	10000	0.93	103	*INSERT*	1	784	809	788		
	10400	0.96	102	*INSERT*	1	784	801	788		
	10400	0.95	100	*SWAP*	1	784	809	789		
	11200	0.90	101	*SWAP*	1	784	813	788		
la18	10600	0.92	100	*INSERT*	1	848	911	858	848	848
	10500	0.87	98	*INSERT*	1	848	908	856		
	10500	0.88	98	*INSERT*	1	848	895	856		
	10500	0.95	98	*INSERT*	1	848	930	857		
	10400	0.89	101	*INSERT*	1	848	912	856		
	10400	0.82	105	*INSERT*	1	848	916	854		
	10300	0.92	100	*INSERT*	1	848	909	859		
	10300	0.91	97	*SWAP*	1	848	898	855		
	10100	0.92	97	*SWAP*	1	852	901	859		
	10100	0.80	104	*INSERT*	1	848	924	853		
	9700	0.85	103	*INSERT*	1	848	890	855		

Instância	Parâmetros					Soluções (20 execuções)			Melhor Solução Protótipo	Solução Ótima (Literatura)
	TI	CR	EL	EV	Freq.	Min	Máx	Média (10 melhores)		
la19	12000	0.93	102	*INSERT*	1	842	900	865	848	842
	11200	0.87	103	*INSERT*	1	842	907	862		
	11000	0.95	97	*INSERT*	1	852	907	863		
	10900	0.86	100	*SWAP*	1	842	900	865		
	10800	0.89	104	*SWAP*	1	850	911	860		
	10700	0.91	101	*INSERT*	1	852	907	859		
	10600	0.88	105	*SWAP*	1	852	909	866		
	10500	0.93	99	*SWAP*	1	850	919	859		
	10500	0.85	102	*SWAP*	1	842	927	860		
	9800	0.80	105	*INSERT*	1	850	905	859		
la20	10900	0.86	101	SWAP	4	907	988	912	907	902
	10900	0.88	101	SWAP	3	907	969	914		
	10900	0.89	101	SWAP	2	907	987	913		
	10800	0.88	100	INSERT	2	907	990	914		

4.3. Discussão de Resultados

O processo de parametrização afeta significativamente o desempenho da Meta-Heurística, logo é crucial perceber como é que este processo pode ser otimizado de forma a permitir que a técnica de otimização consiga obter a melhor solução de forma eficiente e eficaz. Neste trabalho, o processo de parametrização foi abordado através da proposta de um sistema que utiliza os princípios que caracterizam o RL para parametrizar uma Meta-Heurística. A Meta-Heurística é responsável por resolver o Problema de Escalonamento em *Job-Shop*. Em função do desempenho obtido com cada configuração, o Agente é responsável por controlar o valor de cada parâmetro.

Para analisar a exequibilidade do *Framework* de Parametrização, foi implementado um Protótipo para parametrizar o SA. Quando se analisa o impacto que um plano de Escalonamento pode ter, a qualidade do mesmo é um fator essencial. Neste trabalho, o objetivo foi definir um processo de treino que permita ao Agente aprender uma política que lhe permita encontrar sucessivamente a melhor solução para uma determinada instância, isto é, o melhor plano de Escalonamento. Na resolução de 20 instâncias propostas por Lawrence, os resultados evidenciaram que o Agente é capaz de encontrar soluções de qualidade, obtendo inclusive a melhor solução em 17 instâncias. Nas restantes, a diferença não foi de todo significativa, sendo de apenas 1, 5 e 6 unidades temporais (em termos de *makespan*).

Considerando que os intervalos de valores de cada parâmetro foram definidos de acordo com a informação disponível e recomendada na literatura, e conduzem a soluções boas, próximas das melhores para uma determinada instância de um problema, o processo de treino/pesquisa foi modelado para o Agente não só identificar a melhor solução uma vez, mas fazê-lo sucessivamente, ao longo de vários episódios, e, de preferência, no número de iterações mínimo possível, dentro das respetivas circunstâncias. Os registos das evoluções do número de iterações executadas, do melhor *makespan* e do *epsilon* por episódio, para todas as instâncias que o Agente foi capaz de identificar a melhor solução, evidenciam a capacidade de aprendizagem do Agente. De forma geral, os resultados mostram que o Agente é capaz de encontrar a melhor solução várias vezes durante a fase de exploração, perceber quais é que conduzem à melhor solução repetidamente, ou seja, aquelas com uma qualidade superior, e, quando é desafiado a colocar o seu conhecimento em prática, supera o desafio com sucesso, encontrando-a num número de iterações significativamente inferior ao que caracteriza as fases anteriores do processo de pesquisa. Quando, para uma determinada instância, o espaço de soluções é mais irregular, possivelmente com a presença de muitos ótimos locais, aumenta a dificuldade do Agente para encontrar a melhor solução sucessivamente. O Agente é capaz de definir a configuração de parâmetros que conduz a Meta-Heurística ao melhor valor, no entanto, numa fase posterior, quando volta a esta configuração, pode obter resultados diferentes, o que prejudica a sua capacidade de aprendizagem. Nestes casos, a análise dos estados visitados e das ações tomadas evidenciam que o Agente opta por estas combinações repetidamente até que elas sejam capazes de obter a melhor solução.

Para analisar a verdadeira qualidade de cada configuração de parâmetros que o Agente identificou repetidamente como sendo a melhor, a Meta-Heurística foi executada, de forma isolada, para resolver cada instância. Nas situações onde o Agente foi capaz de identificar combinações de parâmetros com uma frequência significativa tendo em conta o número de episódios analisados, a média das 10 melhores soluções foram capazes de confirmar a qualidade de cada configuração, na medida em que resultam no melhor valor reportado na literatura. Por sua vez, nas situações onde a diversidade de resultados é superior, em todas as instâncias a melhor solução encontrada pelo Protótipo corresponde à melhor solução reportada, no entanto, a média das três melhores é ligeiramente diferente. Neste caso, estes resultados sugerem que, possivelmente, a Meta-Heurística não teve tempo suficiente para explorar de forma completa e necessária o espaço de soluções, o que significa que não foi capaz de encontrar uma ou várias combinações de parâmetros que produzam consistentemente os mesmos resultados para o valor do número de iterações do Critério de Interrupção do SA definido. Apesar disto, o facto do Agente, no momento onde o *epsilon* é reduzido, ser capaz de priorizar combinações de parâmetros que demonstraram ser boas num determinado momento e construir um caminho que lhe permita manter ou voltar a testá-las, evidencia a qualidade do sistema proposto e a sua capacidade para obter resultados positivos nestas instâncias.

5. Conclusão

O Capítulo 5, intitulado “Conclusão”, sintetiza as principais conclusões resultantes da análise e discussão dos resultados obtidos com a resolução de 20 instâncias pelo Protótipo do *Framework* de Parametrização de Meta-Heurísticas. De seguida, são discutidas as limitações enfrentadas ao longo da investigação, sejam elas de ordem metodológica, de recursos ou relacionadas com a abrangência dos resultados. Por fim, são sugeridas propostas que possam dar continuidade ao presente estudo e permitam explorar novas abordagens, mitigar as limitações identificadas e/ou ampliar o escopo de aplicação.

5.1. Conclusões Finais

O Escalonamento das Operações é frequentemente um problema crítico em diversos setores, onde a alocação eficiente dos recursos às operações e a sua distribuição no tempo são essenciais para maximizar a produtividade e minimizar custos. A resolução destes problemas, em vários momentos, apresenta uma elevada complexidade devido ao número de variáveis e restrições envolvidas, o que torna a procura por soluções ótimas num desafio considerável. Para enfrentar esses desafios, recorrer a técnicas que envolvam a enumeração das soluções pode ser, em muitos casos, impraticável, porque, em certos ambientes, a dimensão do problema aumenta significativamente, o que tem como consequência um aumento exponencial do número de soluções. Como alternativa, surgem as técnicas de otimização, como Meta-Heurísticas, que têm sido amplamente utilizadas porque, embora não encontrem sempre a solução ótima, elas apresentam a capacidade de encontrar soluções viáveis para problemas complexos em tempos computacionais aceitáveis.

Na sua generalidade, as Meta-Heurísticas são técnicas poderosas que exploram o espaço de soluções de forma eficiente. Contudo, o desempenho dessas técnicas depende fortemente da parametrização adequada de seus parâmetros. A definição dos seus valores é crucial porque são os parâmetros da Meta-Heurística que vão guiar o processo de pesquisa, equilibrar os conceitos de intensidade e diversidade do espaço de soluções, o que faz com que seja impossível desassociar o desempenho de uma Meta-Heurística da parametrização. A escolha dos parâmetros corretos pode significar a diferença entre uma solução subótima e uma solução que se aproxima do ótimo global. No entanto, a parametrização exige tradicionalmente um esforço considerável e um conhecimento especializado, tornando o processo demorado e suscetível a erros. Diante desse cenário, surge a necessidade de métodos automáticos e

adaptativos para a parametrização de Meta-Heurísticas, capazes de ajustar dinamicamente os parâmetros conforme o problema a ser resolvido.

Este trabalho surge com o propósito de perceber se é possível e viável aplicar técnicas de ML para parametrizar Meta-Heurísticas, de forma a simplificar e automatizar todo o processo de definição dos valores de cada parâmetro, melhorar o processo de interação de um utilizador com o mesmo e ultrapassar os problemas que as técnicas atuais de parametrização enfrentam. Sendo assim, foi desenvolvido um Protótipo de um *Framework* de Parametrização que utiliza técnicas de ML para parametrizar o SA. Após a leitura de uma instância, um Agente decide o valor de cada parâmetro, representado através de uma configuração de parâmetros, comunica com a Meta-Heurística, e esta é responsável por a utilizar para resolver o problema. Com base no desempenho de cada configuração, o Agente decide como ajustar cada parâmetro e, num processo de constante troca de informação, o objetivo é que seja capaz de identificar a configuração de parâmetros capaz de permitir à Meta-Heurística obter a melhor solução.

O trabalho desenvolvido nesta dissertação confirma a possibilidade de utilizar técnicas de ML para parametrizar Meta-Heurísticas através do desenvolvimento de um Agente que é capaz de explorar os diferentes estados e ações para identificar a configuração de parâmetros que permite à Meta-Heurística obter a melhor solução para uma determinada instância ou problema. O Agente utiliza os princípios que caracterizam o RL para alcançar os seus objetivos. Neste caso, o algoritmo *Q-Learning* é utilizado para armazenar a informação recolhida durante o processo de exploração e serve de base para as ações tomadas em cada momento. Por sua vez, o mecanismo de recompensa assume um destaque crucial porque influencia o modo como o Agente interpreta o desempenho de cada configuração de parâmetros da Meta-Heurística.

O Protótipo do *Framework* de Parametrização demonstrou ser um sistema robusto e eficaz na resolução de 20 instâncias representativas de um Problema de Escalonamento em *Job-Shop*, especialmente na Minimização do Tempo Total de Conclusão, que é o *makespan*, sendo capaz de encontrar a melhor solução em 17 instâncias, e, nas restantes 3, a melhor solução aproxima-se significativamente da melhor solução documentada na literatura, com diferenças mínima de apenas 1, 5 e 6 unidades temporais em instâncias com a dimensão 10x10, que é a maior dimensão entre as analisadas.

Quando se analisa o efeito de aprendizagem do Agente, bem como a qualidade das combinações de parâmetros que permitiram à Meta-Heurística alcançar a melhor solução, é possível identificar comportamentos diferentes, que são motivados pela dimensão da instância e pelas características do espaço de soluções. Em instâncias de menor dimensão, como a la01 e la05 a la15, onde o espaço de soluções é mais uniforme, o sistema desenvolvido demonstrou uma robustez notável. O Agente, ao longo do processo de resolução, foi consistentemente capaz de encontrar a melhor solução em várias execuções. Independentemente do ponto de partida do algoritmo, observa-se uma tendência clara de convergência para uma região específica do espaço de soluções, onde as soluções de qualidade estão concentradas. Esta característica não só reflete a robustez do sistema, mas também a sua eficiência, pois o Agente não só foi capaz de encontrar a melhor solução, como adquiriu a capacidade de o fazer num número reduzido de iterações porque conseguiu identificar configurações de parâmetros de qualidade. Quando o espaço de soluções é mais irregular, como nas instâncias la02 e la04, a

dificuldade para o sistema encontrar a melhor solução consistentemente, num número de iterações "adequado", aumenta. O sistema consegue alcançar a melhor solução, identifica a configuração (ou configurações) de parâmetros que permitiram isso, e tende a revisitar o(s) estado(s). No entanto, em vários momentos, a mesma configuração de parâmetros não produz consistentemente os mesmos resultados. Este comportamento acentua-se em instâncias de dimensão superior, como as la16 a la20, mas, nestas situações, a dificuldade para encontrar a melhor solução várias vezes ainda se acentua de forma mais significativa, o que é motivado pela natureza estocástica das Meta-Heurísticas, mas também pelo número de iterações do Critério de Interrupção da Meta-Heurística definido, que, para os problemas em questão, necessita de ser significativamente elevado para se alcançar uma variabilidade muito reduzida na qualidade das soluções. Neste trabalho, a escolha deste valor para o Critério de Interrupção da Meta-Heurística foi balanceada por limitações práticas de tempo e recursos computacionais. O processo de treino envolve ciclos contínuos de ajuste e avaliação, onde o tempo de execução é um fator crítico. Aumentar o Critério de Interrupção prolongaria significativamente o tempo de cada iteração do SA, tornando o processo de otimização mais custoso e demorado. Dessa forma, a escolha do valor utilizado reflete um compromisso entre a necessidade de uma exploração eficaz e as restrições de tempo impostas pelo contexto experimental. Mesmo assim, os resultados obtidos permitem concluir que, os princípios que caracterizam este sistema, apresentam um potencial elevado para obter bons resultados nas várias instâncias, caso seja possível aumentar o Critério de Interrupção para valores mais adequados.

A utilização de ML na parametrização de Meta-Heurísticas representa assim uma abordagem inovadora e poderosa no campo da otimização. Ao explorar a capacidade do RL para aprender e adaptar-se a diferentes cenários, o modelo desenvolvido neste trabalho demonstra uma melhoria significativa na eficácia e eficiência das Meta-Heurísticas. A automatização da parametrização resulta em soluções robustas e capazes de lidar com a variabilidade e a complexidade dos Problemas de Otimização.

Numa vertente direcionada para o desempenho das técnicas de otimização, este trabalho contribui para a redução do esforço necessário para configurar Meta-Heurísticas. Além disso, também proporciona um aumento do desempenho dos algoritmos de otimização ao ajustar de forma contínua e inteligente os parâmetros, maximizando a exploração do espaço de soluções. A combinação de RL com Meta-Heurísticas não apenas assegura uma boa qualidade nas soluções encontradas, mas também amplia o potencial de aplicação dessas técnicas em problemas complexos e dinâmicos, onde a parametrização tradicional pode ser inadequada ou impraticável.

Em termos de contribuições, mas relacionadas com a demonstração do conteúdo e resultados obtidos neste trabalho, podem ser mencionadas as seguintes publicações e premiações obtidas:

→ 1 Publicação como autor principal: D. Dias, A. S. Santos, and L. R. Varela, "Machine Learning Algorithms in Scheduling Problems: An Overview and Future Paths," in *Innovations in Mechatronics Engineering III*, J. Machado, F. Soares, J. Trojanowska, S. Yildirim, J. Vojtěšek, P. Rea, B. Gramescu, and O. O. Hrybiuk, Eds., Cham: Springer Nature Switzerland, 2024, pp. 79–89 [292];

- → Prémio de Mestrado OERN – 50 prémios para alunos de 2º ano dos cursos de Mestrado de Engenharia lecionados em Instituições de Ensino Superior Público da Região Norte;
- → Publicação: B. Silva, F. Ferreira, G. Magalhães, I. Azevedo, D. Dias, A. S. Santos, and L. R. Varela (2024). Literature Review of Artificial Intelligence Applications in Production Planning and Scheduling (Submitted);
- → Publicação em curso: Artigo de Revista focado na apresentação das características do Protótipo de Parametrização proposto neste trabalho e nos resultados obtidos no estudo computacional realizado.

5.2. Limitações e Trabalhos Futuros

Durante a realização deste trabalho, as diversas limitações enfrentadas relacionam-se com a complexidade das diferentes técnicas e materiais utilizados para desenvolver o Protótipo do *Framework* de Parametrização, que, de certa forma, são comuns em projetos que envolvem o uso de IA e/ou técnicas de otimização. Uma das limitações principais relaciona-se com a qualidade do hardware disponível e usado, que desempenha um papel crucial em trabalhos de ML, onde o poder computacional é determinante para o desempenho do sistema. No caso deste estudo, o equipamento utilizado para realizar o estudo computacional é consideravelmente modesto, o que possivelmente impactou o desempenho do Agente. Em particular, para instâncias de maiores dimensões, uma máquina mais potente poderia ter permitido aumentar significativamente o número de iterações por episódio durante o processo de pesquisa, possibilitando assim uma fase de exploração mais longa, completa e eficaz, e, consequentemente, o alcance de resultados potencialmente com qualidade superior.

Complementar à qualidade do computador, é importante destacar a natureza estocástica das Meta-Heurísticas. Devido a essa característica, a mesma configuração de parâmetros pode produzir resultados ligeiramente diferentes em execuções distintas. Para mitigar essa variabilidade, cada configuração de parâmetros foi testada cinco vezes, e a média dos três melhores resultados foi considerada como representativa do desempenho. No entanto, à medida que a complexidade das instâncias aumenta, a consistência dos resultados tende a diminuir. Mesmo após várias execuções, a média obtida pode não ser replicada em futuras iterações. Essa variação torna-se particularmente crítica quando o Agente decide reutilizar uma configuração de parâmetros que, em um momento anterior, produziu a melhor solução. Nessas circunstâncias, é comum observar que, embora a configuração tenha sido eficaz em um cenário, os resultados subsequentes podem apresentar flutuações, o que destaca os desafios adicionais associados à otimização em ambientes estocásticos e de alta complexidade.

Para trabalhos futuros, várias direções de pesquisa podem ser exploradas para ampliar e aprimorar os resultados obtidos neste estudo. Uma linha promissora seria a aplicação do Protótipo do *Framework* de Parametrização desenvolvido em instâncias com uma complexidade diferente. A complexidade crescente dessas instâncias representa um desafio significativo, mas também oferece uma oportunidade para testar a escalabilidade e a robustez do Agente, bem como da Meta-heurística. Investigar como o Agente se comporta em contextos

mais complexos poderia fornecer insights valiosos sobre a necessidade de ajustes adicionais no processo de treino e na estrutura do *Q-Learning*. Como complemento à resolução das instâncias apresentadas neste trabalho, também seria interessante utilizar o processo de treino implementado, mas com um número de iterações do Critério de Interrupção da Meta-Heurística mais elevado para comprovar os resultados enunciados esperados.

Além disso, embora tenham sido testadas várias combinações de parâmetros que caracterizam o algoritmo de *Q-Learning*, há sempre a possibilidade de explorar configurações adicionais. O estudo sistemático de diferentes valores para parâmetros como a taxa de aprendizagem, o fator de desconto, e o *epsilon* na política *epsilon-greedy*, poderia revelar maneiras de aprimorar o desempenho do Agente. Esse aprofundamento poderia ajudar a compreender melhor o impacto de cada parâmetro individualmente e em conjunto, permitindo uma otimização mais refinada do processo de aprendizagem.

Noutra vertente, poderia ser interessante utilizar os princípios que caracterizam o *Framework* de Parametrização e implementar outras Meta-Heurísticas com ampla aplicação como o TS, o *Artificial Bee Colony, Genetic Algorithms*, ou Meta-Heurísticas mais recentes, como *Grey Wolf Optimizer* e *Spider Wasp Optimizer*. Neste último caso, o objetivo também seria perceber como é que as mesmas se comportariam, uma vez que o estudo sobre as mesmas ainda não é tão aprofundado como acontece com as Meta-Heurísticas mais clássicas.

Por fim, embora este estudo tenha alcançado seus objetivos principais, demonstrando a viabilidade de um Agente Inteligente para a Parametrização de Meta-Heurísticas em Problemas de Escalonamento, existe sempre espaço para melhorias e novas abordagens. A abertura para a exploração de técnicas complementares, tanto a nível de algoritmo de ML como de técnicas de otimização, e a adaptação do Agente a problemas de maior dimensão são apenas algumas das direções em que o trabalho pode ser expandido. Dessa forma, o presente estudo estabelece uma base sólida, mas deixa em aberto a possibilidade de novas contribuições que possam ampliar ainda mais o campo de aplicação e os resultados obtidos.

Referências

[1] Z. Jiang, S. Yuan, J. Ma, and Q. Wang, "The evolution of production scheduling from Industry 3.0 through Industry 4.0," *Int J Prod Res*, vol. 60, no. 11, pp. 3534–3554, Jun. 2022, doi: 10.1080/00207543.2021.1925772.

[2] K. Sörensen, "Metaheuristics-the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, 2015, doi: 10.1111/itor.12001.

[3] K. Hussain, M. N. Mohd Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: a comprehensive survey," *Artif Intell Rev*, vol. 52, no. 4, pp. 2191–2233, 2019, doi: 10.1007/s10462-017-9605-z.

[4] Saunders. Mark N. K., P. Lewis, and A. Thornhill, *Research Methods for Business Students*, 8th ed. Pearson Education, 2019.

[5] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.

[6] W. Kuechler and V. Vaishnavi, "A Framework for Theory Development in Design Science Research: Multiple Perspectives," *Journal of the Association of Information Systems*, vol. 13, pp. 395–423, Jun. 2012, doi: 10.17705/1jais.00300.

[7] J. F. Beasley, "OR-Library."

[8] C. Wohlin, "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, in EASE '14. New York, NY, USA: Association for Computing Machinery, 2014. doi: 10.1145/2601248.2601268.

[9] J. Y. T. Leung, *Handbook of scheduling: Algorithms, models, and performance analysis*, 1st ed. New York, USA: Chapman and Hall/CRC, 2004. doi: https://doi.org/10.1201/9780203489802.

[10] M. E. Salveson, "On a Quantitative Method in Production Planning and Scheduling," *Econometrica*, vol. 20, no. 4, pp. 554–590, 1952, doi: 10.2307/1907643.

[11] I. Harjunkoski *et al.*, "Scope for industrial applications of production scheduling models and solution methods," 2014. doi: 10.1016/j.compchemeng.2013.12.001.

[12] A. Santos, "Auto-parametrização de meta-heurísticas para problemas de escalonamento em ambiente industrial," Ph.D. Dissertation, Universidade do Minho, Guimarães, Portugal, 2020.

[13] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 5th ed. New York, USA: Springer, 2016.

[14] P. Brucker, *Scheduling Algorithms*. Springer Berlin, Heidelberg, 2007. doi: https://doi.org/10.1007/978-3-540-69516-5.

[15] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," 2009. doi: 10.1007/s10951-008-0090-8.

[16] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, “Automated Design of Production Scheduling Heuristics: A Review,” *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, 2016, doi: 10.1109/TEVC.2015.2429314.

[17] D. A. Rossit, F. Tohmé, and M. Frutos, “Industry 4.0: Smart Scheduling,” *Int J Prod Res*, vol. 57, no. 12, 2019, doi: 10.1080/00207543.2018.1504248.

[18] K. R. Baker and D. Trietsch, *Principles of Sequencing and Scheduling*, 2nd ed. New Jersey, EUA: John Wiley & Sons, Inc., 2018. doi: 10.1002/9781119262602.

[19] A. Artiba and S. E. Elmaghraby, *The Planning and Scheduling of Production Systems*. Berlin, Germany: Springer Science & Business Media, 1996.

[20] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, M. Sterna, and J. Weglarz, *Handbook on Scheduling: From Theory to Practice*, 2nd ed. Cham, Switzerland: Springer International Publishing, 2019. doi: 10.1007/978-3-319-99849-7.

[21] A. Santos, “Análise do desempenho de técnicas de otimização no problema de escalonamento,” M.S. Thesis, Universidade do Minho, Guimarães, Portugal, 2015.

[22] M. L. R. Varela, “Uma contribuição para o escalonamento da produção baseado em métodos globalmente distribuídos,” Ph.D. Dissertation, Universidade do Minho, Guimarães, Portugal, 2007.

[23] L. W. Conway, R. W., Maxwell, W. L., Miller, *Theory of Scheduling (Dover Books on Computer Science)*. Dover Publications, 1967.

[24] A. H. G. Graham, R. L., Lawler, E. L., Lenstra, J. K. & Rinnooy Kan, “Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey,” *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[25] J. Chen, J. Peng, and D. K. J. Lin, “A statistical perspective on non-deterministic polynomial-time hard ordering problems: Making use of design for order-of-addition experiments,” *Comput Ind Eng*, vol. 162, no. October, p. 107773, 2021, doi: 10.1016/j.cie.2021.107773.

[26] E. G. Talbi, *Metaheuristics: From Design to Implementation*. New Jersey, USA: John Wiley & Sons, 2009.

[27] J. Gmys, M. Mezmaz, N. Melab, and D. Tuyttens, “A computationally efficient Branch-and-Bound algorithm for the permutation flow-shop scheduling problem,” *Eur J Oper Res*, vol. 284, no. 3, pp. 814–833, 2020, doi: 10.1016/j.ejor.2020.01.039.

[28] C. P. Tomazella and M. S. Nagano, “A comprehensive review of Branch-and-Bound algorithms: Guidelines and directions for further research on the flowshop scheduling problem,” *Expert Syst Appl*, vol. 158, p. 113556, 2020, doi: 10.1016/j.eswa.2020.113556.

[29] J. Ahn and H. J. Kim, “A Branch and Bound Algorithm for Scheduling of Flexible Manufacturing Systems,” *IEEE Transactions on Automation Science and Engineering*, vol. PP, pp. 1–15, 2023, doi: 10.1109/TASE.2023.3296087.

[30] R. Bellman, *Dynamic Programing*. Princeton, New Jersey: Princeton University Press, 1957.

[31] H. Rezaei, S. E. Abdollahi, S. Abdollahi, and S. Filizadeh, “Energy management strategies of battery-ultracapacitor hybrid storage systems for electric vehicles: Review, challenges, and future trends,” *J Energy Storage*, vol. 53, no. December 2021, p. 105045, 2022, doi: 10.1016/j.est.2022.105045.

[32] E. A. G. de Souza, M. S. Nagano, and G. A. Rolim, “Dynamic Programming algorithms and their applications in machine scheduling: A review,” *Expert Syst Appl*, vol. 190, no. March 2021, p. 116180, 2022, doi: 10.1016/j.eswa.2021.116180.

[33] P. Yunusoglu and S. Topaloglu Yildiz, “Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-

dependent setup times," *Int J Prod Res*, vol. 60, no. 7, pp. 2212–2229, Apr. 2022, doi: 10.1080/00207543.2021.1885068.
[34] F. Rossi, P. Van Beek, and T. Walsh Elsevier, *Handbook of Constraint Programming*, 1st ed. Elsevier, 2006.
[35] L. Meng, C. Zhang, Y. Ren, B. Zhang, and C. Lv, "Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem," *Comput Ind Eng*, vol. 142, 2020, doi: 10.1016/j.cie.2020.106347.
[36] X. He, Y. Yao, Z. Chen, J. Sun, and H. Chen, "Efficient parallel A* search on multi-GPU system," *Future Generation Computer Systems*, vol. 123, 2021, doi: 10.1016/j.future.2021.04.011.
[37] E. Burns and W. Ruml, "Iterative-deepening search with on-line tree size prediction," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012. doi: 10.1007/978-3-642-34413-8_1.
[38] W. Zhu, H. Qin, A. Lim, and H. Zhang, "Iterative deepening A* algorithms for the container relocation problem," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 4, 2012, doi: 10.1109/TASE.2012.2198642.
[39] H. Y. Fuchigami and S. Rangel, "A survey of case studies in production scheduling: Analysis and perspectives," *J Comput Sci*, vol. 25, 2018, doi: 10.1016/j.jocs.2017.06.004.
[40] K. Genova and D. P. Williamson, "An Experimental Evaluation of the Best-of-Many Christofides' Algorithm for the Traveling Salesman Problem," *Algorithmica*, vol. 78, no. 4, pp. 1109–1130, 2017, doi: 10.1007/s00453-017-0293-5.
[41] Á. Felipe, M. T. Ortuño, G. Righini, and G. Tirado, "A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges," *Transp Res E Logist Transp Rev*, vol. 71, 2014, doi: 10.1016/j.tre.2014.09.003.
[42] Q. K. Pan, L. Gao, L. Wang, J. Liang, and X. Y. Li, "Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem," *Expert Syst Appl*, vol. 124, 2019, doi: 10.1016/j.eswa.2019.01.062.
[43] G. Da Col and E. C. Teppan, "Industrial-size job shop scheduling with constraint programming," *Operations Research Perspectives*, vol. 9, p. 100249, 2022, doi: https://doi.org/10.1016/j.orp.2022.100249.
[44] C. Rajendran and O. Holthaus, "Comparative study of dispatching rules in dynamic flowshops and jobshops," *Eur J Oper Res*, vol. 116, no. 1, 1999, doi: 10.1016/S0377-2217(98)00023-X.
[45] M. Đurasević and D. Jakobović, "A survey of dispatching rules for the dynamic unrelated machines environment," *Expert Syst Appl*, vol. 113, pp. 555–569, 2018, doi: https://doi.org/10.1016/j.eswa.2018.06.053.
[46] F. Peres and M. Castelli, "Combinatorial Optimization Problems and Metaheuristics: Review, Challenges, Design, and Development," *Applied Sciences*, vol. 11, no. 14, 2021, doi: 10.3390/app11146449.
[47] T. Van Luong, N. Melab, and E.-G. Talbi, "GPU Computing for Parallel Local Search Metaheuristics," *IEEE Trans. Computers*, vol. 62, pp. 173–185, Jan. 2013, doi: 10.1109/TC.2011.206.
[48] C. Bierwirth, D. Mattfeld, and H. Kopfer, "On Permutation Representations for Scheduling Problems," *Proceedings of Parallel Problem Solving from Nature*, vol. 4, Aug. 1997, doi: 10.1007/3-540-61723-X_995.
[49] J. Xie, X. Li, L. Gao, and L. Gui, "A new neighborhood structure for job shop scheduling problems," *CoRR*, vol. abs/2109.02843, 2021, [Online]. Available: https://arxiv.org/abs/2109.02843

[50] A. S. Santos, A. M. Madureira, and M. L. R. Varela, “The Influence of Problem Specific Neighborhood Structures in Metaheuristics Performance,” *Journal of Mathematics*, vol. 2018, p. 8072621, 2018, doi: 10.1155/2018/8072621.

[51] K. Tamssaouet and S. Dauzère-Pérès, “A general efficient neighborhood structure framework for the job-shop and flexible job-shop scheduling problems,” *Eur J Oper Res*, vol. 311, no. 2, pp. 455–471, 2023, doi: https://doi.org/10.1016/j.ejor.2023.05.018.

[52] M. Besten, “Neighborhoods Revisited: An Experimental Investigation into the Effectiveness of Variable Neighborhood Descent for Scheduling,” Aug. 2001.

[53] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, “Job Shop Scheduling by Simulated Annealing,” *Oper Res*, vol. 40, no. 1, pp. 113–125, 1992, [Online]. Available: http://www.jstor.org/stable/171189

[54] M. Şevkli and M. Aydin, “Variable Neighbourhood Search for Job Shop Scheduling Problems,” *JSW*, vol. 1, pp. 34–39, Aug. 2006, doi: 10.4304/jsw.1.2.34-39.

[55] Q. Luo, J. Xie, M. Ma, and L. Li, “Discrete Bat Algorithm for Optimal Problem of Permutation Flow Shop Scheduling,” *The Scientific World Journal*, vol. 2014, p. 630280, Aug. 2014, doi: 10.1155/2014/630280.

[56] C. B. Kalayci, O. Polat, and S. M. Gupta, “A variable neighbourhood search algorithm for disassembly lines,” *Journal of Manufacturing Technology Management*, vol. 26, no. 2, pp. 182–194, Jan. 2015, doi: 10.1108/JMTM-11-2013-0168.

[57] H. Liu and L. Zhang, “Optimizing a Disassembly Sequence Planning With Success Rates of Disassembly Operations via a Variable Neighborhood Search Algorithm,” *IEEE Access*, vol. 9, pp. 157540–157549, 2021, doi: 10.1109/ACCESS.2021.3101221.

[58] H. F. Amaral, S. Urrutia, and L. M. Hvattum, “Delayed improvement local search,” *Journal of Heuristics*, vol. 27, no. 5, pp. 923–950, 2021, doi: 10.1007/s10732-021-09479-9.

[59] F. Xhafa and A. Abraham, *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*, vol. 128. Springer Berlin, Heidelberg, 2008. doi: 10.1007/978-3-540-78985-7.

[60] K. Z. Gao, P. N. Suganthan, Q. K. Pan, T. J. Chua, T. X. Cai, and C. S. Chong, “Pareto-based grouping discrete harmony search algorithm for multi-objective flexible job shop scheduling,” *Inf Sci (N Y)*, vol. 289, no. 1, 2014, doi: 10.1016/j.ins.2014.07.039.

[61] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, “A survey on metaheuristics for stochastic combinatorial optimization,” *Nat Comput*, vol. 8, no. 2, 2009, doi: 10.1007/s11047-008-9098-4.

[62] A. Kaveh, *Advances in Metaheuristic Algorithms for Optimal Design of Structures*, 3th ed. Springer Cham, 2021. doi: 10.1007/978-3-030-59392-6.

[63] X.-S. Yang, *Nature-Inspired Optimization Algorithms*, 2nd ed. Cambridge, United States: Academic Press, 2020. doi: 10.1016/B978-0-12-821986-7.00002-0.

[64] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Comput Oper Res*, vol. 13, no. 5, 1986, doi: 10.1016/0305-0548(86)90048-1.

[65] I. H. Osman and G. Laporte, “Metaheuristics: A bibliography,” *Ann Oper Res*, vol. 63, 1996, doi: 10.1007/bf02125421.

[66] S. Voß, S. Martello, I. H. Osman, and C. Roucairol, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, 1th ed. New York, NY: Springer Science+Business Media, 1999. doi: 10.1007/978-1-4615-5775-3.

[67] C. Blum and A. Roli, “Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison,” 2003. doi: 10.1145/937503.937505.

[68] S. Mirjalili, S. M. Mirjalili, and A. Lewis, “Grey Wolf Optimizer,” *Advances in Engineering Software*, vol. 69, 2014, doi: 10.1016/j.advengsoft.2013.12.007.

[69] B. Morales-Castañeda, D. Zaldívar, E. Cuevas, F. Fausto, and A. Rodríguez, “A better balance in metaheuristic algorithms: Does it exist?,” *Swarm Evol Comput*, vol. 54, p. 100671, 2020, doi: https://doi.org/10.1016/j.swevo.2020.100671.

[70] M. J. Blondin, “Controller Tuning by Metaheuristics Optimization,” in *Controller Tuning Optimization Methods for Multi-Constraints and Nonlinear Systems: A Metaheuristic Approach*, Cham, Switzerland: Springer International Publishing, 2021, pp. 11–51. doi: 10.1007/978-3-030-64541-0_2.

[71] A. E. Ezugwu *et al.*, “Metaheuristics: a comprehensive overview and classification along with bibliometric analysis,” *Artif Intell Rev*, vol. 54, no. 6, pp. 4237–4316, 2021, doi: 10.1007/s10462-020-09952-0.

[72] I. Boussaïd, J. Lepagnot, and P. Siarry, “A survey on optimization metaheuristics,” *Inf Sci (N Y)*, vol. 237, pp. 82–117, 2013, doi: https://doi.org/10.1016/j.ins.2013.02.041.

[73] K. Rajwar, K. Deep, and S. Das, “An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges,” *Artif Intell Rev*, vol. 56, no. 11, pp. 13187–13257, 2023, doi: 10.1007/s10462-023-10470-y.

[74] R. Rajakumar, P. Dhavachelvan, and T. Vengattaraman, “A survey on nature inspired meta-heuristic algorithms with its domain specifications,” in *2016 International Conference on Communication and Electronics Systems (ICCES)*, 2016, pp. 1–6. doi: 10.1109/CESYS.2016.7889811.

[75] A. W. Mohamed, A. A. Hadi, and A. K. Mohamed, “Gaining-sharing knowledge based algorithm for solving optimization problems: a novel nature-inspired algorithm,” *International Journal of Machine Learning and Cybernetics*, vol. 11, no. 7, 2020, doi: 10.1007/s13042-019-01053-x.

[76] D. Molina, J. Poyatos, J. Del Ser, S. García, A. Hussain, and F. Herrera, “Comprehensive Taxonomies of Nature- and Bio-inspired Optimization: Inspiration Versus Algorithmic Behavior, Critical Analysis Recommendations,” *Cognit Comput*, vol. 12, no. 5, pp. 897–939, 2020, doi: 10.1007/s12559-020-09730-8.

[77] H. Stegherr, M. Heider, and J. Hähner, “Classifying Metaheuristics: Towards a unified multi-level classification system,” *Nat Comput*, vol. 21, no. 2, 2022, doi: 10.1007/s11047-020-09824-0.

[78] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science (1979)*, vol. 220, no. 4598, 1983, doi: 10.1126/science.220.4598.671.

[79] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*. 1987. doi: 10.1007/978-94-015-7744-1.

[80] S. and M. M. Delahaye Daniel and Chaimatanan, “Simulated Annealing: From Basics to Applications,” in *Handbook of Metaheuristics*, J.-Y. Gendreau Michel and Potvin, Ed., Cham: Springer International Publishing, 2019, pp. 1–35. doi: 10.1007/978-3-319-91086-4_1.

[81] M. M. Mafarja and S. Mirjalili, “Hybrid Whale Optimization Algorithm with simulated annealing for feature selection,” *Neurocomputing*, vol. 260, 2017, doi: 10.1016/j.neucom.2017.04.053.

[82] M.-W. Park and Y.-D. Kim, “A systematic procedure for setting parameters in simulated annealing algorithms,” *Comput Oper Res*, vol. 25, no. 3, pp. 207–217, 1998, doi: https://doi.org/10.1016/S0305-0548(97)00054-3.

[83] I. Dupanloup, S. Schneider, and L. Excoffier, “A simulated annealing approach to define the genetic structure of populations,” *Mol Ecol*, vol. 11, no. 12, 2002, doi: 10.1046/j.1365-294X.2002.01650.x.

[84] N. Siddique and H. Adeli, “Simulated Annealing, Its Variants and Engineering Applications,” 2016. doi: 10.1142/S0218213016300015.

[85] E. Triki, Y. Collette, and P. Siarry, “A theoretical study on the behavior of simulated annealing leading to a new cooling schedule,” *Eur J Oper Res*, vol. 166, no. 1, pp. 77–92, 2005, doi: https://doi.org/10.1016/j.ejor.2004.03.035.

[86] F. Romeo and A. Sangiovanni-Vincentelli, “A theoretical framework for simulated annealing,” *Algorithmica*, vol. 6, no. 1, pp. 302–345, 1991, doi: 10.1007/BF01759049.

[87] M. Lundy and A. Mees, “Convergence of an annealing algorithm,” *Math Program*, vol. 34, no. 1, pp. 111–124, 1986, doi: 10.1007/BF01582166.

[88] S. Geman and D. Geman, “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images,” *IEEE Trans Pattern Anal Mach Intell*, vol. PAMI-6, no. 6, pp. 721–741, 1984, doi: 10.1109/TPAMI.1984.4767596.

[89] F. Glover, “Tabu Search: A Tutorial,” *Interfaces (Providence)*, vol. 20, no. 4, 1990, doi: 10.1287/inte.20.4.74.

[90] J. Q. Li, Q. K. Pan, and Y. C. Liang, “An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems,” *Comput Ind Eng*, vol. 59, no. 4, 2010, doi: 10.1016/j.cie.2010.07.014.

[91] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, “Review of job shop scheduling research and its new perspectives under Industry 4.0,” *J Intell Manuf*, vol. 30, no. 4, 2019, doi: 10.1007/s10845-017-1350-2.

[92] C. Y. Zhang, P. G. Li, Z. L. Guan, and Y. Q. Rao, “A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem,” *Comput Oper Res*, vol. 34, no. 11, 2007, doi: 10.1016/j.cor.2005.12.002.

[93] J.-Y. Gendreau Michel and Potvin, “Tabu Search,” in *Handbook of Metaheuristics*, J.-Y. Gendreau Michel and Potvin, Ed., Cham: Springer International Publishing, 2019, pp. 37–55. doi: 10.1007/978-3-319-91086-4_2.

[94] Y. Lu, B. Cao, C. Rego, and F. Glover, “A Tabu search based clustering algorithm and its parallel implementation on Spark,” *Applied Soft Computing Journal*, vol. 63, 2018, doi: 10.1016/j.asoc.2017.11.038.

[95] Y. Alotaibi, “A New Meta-Heuristics Data Clustering Algorithm Based on Tabu Search and Adaptive Search Memory,” *Symmetry (Basel)*, vol. 14, no. 3, 2022, doi: 10.3390/sym14030623.

[96] V. K. Prajapati, M. Jain, and L. Chouhan, “Tabu Search Algorithm (TSA): A Comprehensive Survey,” in *Proceedings of 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things, ICETCE 2020*, 2020. doi: 10.1109/ICETCE48199.2020.9091743.

[97] J. H. Holland, “Genetic Algorithms,” *Sci Am*, vol. 267, no. 1, pp. 66–73, 1992, [Online]. Available: http://www.jstor.org/stable/24939139

[98] S. Katoch, S. S. Chauhan, and V. Kumar, “A review on genetic algorithm: past, present, and future,” *Multimed Tools Appl*, vol. 80, no. 5, pp. 8091–8126, 2021, doi: 10.1007/s11042-020-10139-6.

[99] R. Storn and K. Price, “Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces,” *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997, doi: 10.1023/A:1008202821328.

[100] C. Ferreira, “Gene Expression Programming: a New Adaptive Algorithm for Solving Problems,” *CoRR*, vol. cs.AI/0102027, 2001, [Online]. Available: https://arxiv.org/abs/cs/0102027

[101] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, 1995, pp. 1942–1948 vol.4. doi: 10.1109/ICNN.1995.488968.

[102] D. Wang, D. Tan, and L. Liu, “Particle swarm optimization algorithm: an overview,” *Soft comput*, vol. 22, no. 2, pp. 387–408, 2018, doi: 10.1007/s00500-016-2474-6.

[103] F. Marini and B. Walczak, “Particle swarm optimization (PSO). A tutorial,” *Chemometrics and Intelligent Laboratory Systems*, vol. 149, pp. 153–165, 2015, doi: https://doi.org/10.1016/j.chemolab.2015.08.020.
[104] M. Dorigo, “Optimization, Learning and Natural Algorithms,” *Ph.D. Thesis, Politecnico di Milano*, 1992, Accessed: Dec. 11, 2023. [Online]. Available: https://cir.nii.ac.jp/crid/1573950400977139328.bib?lang=ja
[105] D. Karaboga, “An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report - TR06,” *Technical Report, Erciyes University*, Jan. 2005.
[106] H. Faris, I. Aljarah, M. A. Al-Betar, and S. Mirjalili, “Grey wolf optimizer: a review of recent variants and applications,” *Neural Comput Appl*, vol. 30, no. 2, pp. 413–435, 2018, doi: 10.1007/s00521-017-3272-5.
[107] S. Mirjalili and A. Lewis, “The Whale Optimization Algorithm,” *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016, doi: https://doi.org/10.1016/j.advengsoft.2016.01.008.
[108] M. Abdel-Basset, R. Mohamed, M. Jameel, and M. Abouhawwash, “Spider wasp optimizer: a novel meta-heuristic optimization algorithm,” *Artif Intell Rev*, vol. 56, no. 10, pp. 11675–11738, 2023, doi: 10.1007/s10462-023-10446-y.
[109] M. Kaveh, M. S. Mesgari, and B. Saeidian, “Orchard Algorithm (OA): A new meta-heuristic algorithm for solving discrete and continuous optimization problems,” *Math Comput Simul*, vol. 208, pp. 95–135, 2023, doi: https://doi.org/10.1016/j.matcom.2022.12.027.
[110] A. Alorf, “A survey of recently developed metaheuristics and their comparative analysis,” *Eng Appl Artif Intell*, vol. 117, p. 105622, 2023, doi: https://doi.org/10.1016/j.engappai.2022.105622.
[111] B. Adenso-Díaz and M. Laguna, “Fine-Tuning of Algorithms Using Fractional Experimental Designs and Local Search,” *Oper Res*, vol. 54, no. 1, pp. 99–114, 2006, [Online]. Available: http://www.jstor.org/stable/25146951
[112] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy, “An experimental investigation of model-based parameter optimisation: SPO and beyond,” in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, in GECCO ’09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 271–278. doi: 10.1145/1569901.1569940.
[113] Y. Eryoldaş and A. Durmuşoglu, “A Literature Survey on Offline Automatic Algorithm Configuration,” *Applied Sciences*, vol. 12, no. 13, 2022, doi: 10.3390/app12136316.
[114] S. K. Joshi and J. C. Bansal, “Parameter tuning for meta-heuristics,” *Knowl Based Syst*, vol. 189, p. 105094, 2020, doi: https://doi.org/10.1016/j.knosys.2019.105094.
[115] S. K. Smit and A. E. Eiben, “Comparing parameter tuning methods for evolutionary algorithms,” in *2009 IEEE Congress on Evolutionary Computation*, 2009, pp. 399–406. doi: 10.1109/CEC.2009.4982974.
[116] A. E. Eiben and S. K. Smit, “Evolutionary Algorithm Parameters and Methods to Tune Them,” in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 15–36. doi: 10.1007/978-3-642-21434-9_2.
[117] I. Pereira, A. Madureira, E. Costa E Silva, and A. Abraham, “A Hybrid Metaheuristics Parameter Tuning Approach for Scheduling through Racing and Case-Based Reasoning,” *Applied Sciences*, vol. 11, p. 3325, Apr. 2021, doi: 10.3390/app11083325.
[118] E. S. Skakov and V. N. Malysh, “Parameter meta-optimization of metaheuristics of solving specific NP-hard facility location problem,” *J Phys Conf Ser*, vol. 973, no. 1, p. 12063, Mar. 2018, doi: 10.1088/1742-6596/973/1/012063.

[119] M. Birattari, *Tuning Metaheuristics: A Machine Learning Perspective*, vol. 197. Berlin, Heidelberg: Springer Berlin, Heidelberg, 2009. doi: 10.1007/978-3-642-00483-4.
[120] G. Eiben and M. C. Schut, "New Ways to Calibrate Evolutionary Algorithms," in *Advances in Metaheuristics for Hard Optimization*, P. Siarry and Z. Michalewicz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 153–177. doi: 10.1007/978-3-540-72960-0_8.
[121] A. S. Santos, A. M. Madureira, and L. R. Varela, "A Self-Parametrization Framework for Meta-Heuristics," *Mathematics*, vol. 10, no. 3, 2022, doi: 10.3390/math10030475.
[122] E. Montero, M.-C. Riff, and B. Neveu, "A beginner's guide to tuning methods," *Appl Soft Comput*, vol. 17, pp. 39–51, 2014, doi: https://doi.org/10.1016/j.asoc.2013.12.017.
[123] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm Evol Comput*, vol. 1, no. 1, pp. 19–31, 2011, doi: https://doi.org/10.1016/j.swevo.2011.02.001.
[124] C. Huang, Y. Li, and X. Yao, "A Survey of Automatic Parameter Tuning Methods for Metaheuristics," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 201–216, 2020, doi: 10.1109/TEVC.2019.2921598.
[125] D. C. Montgomery, *Design and Analysis of Experiments*, 10th ed. John Wiley & Sons, Inc., 2019.
[126] L. Eriksson, E. Johansson, and N. Kettaneh-Wold, *Design of Experiments: Principles and Applications*, 3rd ed. Umetrics Academy, Umea, 2008.
[127] M. Amoozegar and E. Rashedi, "Parameter tuning of GSA using DOE," in *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*, 2014, pp. 431–436. doi: 10.1109/ICCKE.2014.6993390.
[128] G. A. Lujan-Moreno, P. R. Howard, O. G. Rojas, and D. C. Montgomery, "Design of experiments and response surface methodology to tune machine learning hyperparameters, with a random forest case-study," *Expert Syst Appl*, vol. 109, pp. 195–205, 2018, doi: https://doi.org/10.1016/j.eswa.2018.05.024.
[129] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil, "Using Experimental Design to Find Effective Parameter Settings for Heuristics," *Journal of Heuristics*, vol. 7, no. 1, pp. 77–97, 2001, doi: 10.1023/A:1026569813391.
[130] F. Dobslaw, "A parameter tuning framework for metaheuristics based on design of experiments and artificial neural networks," *World Acad Sci Eng Technol*, vol. 64, Jan. 2010.
[131] H. C. and L. Gunawan Aldy and Lau, "Fine-Tuning Algorithm Parameters Using the Design of Experiments Approach," in *Learning and Intelligent Optimization*, C. A. C. Coello, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 278–292.
[132] E. B. de M. Barbosa and E. L. F. Senne, "Improving the Fine-Tuning of Metaheuristics: An Approach Combining Design of Experiments and Racing Algorithms," *Journal of Optimization*, vol. 2017, p. 8042436, 2017, doi: 10.1155/2017/8042436.
[133] A. J. Yu and J. Seif, "Minimizing tardiness and maintenance costs in flow shop scheduling by a lower-bound-based GA," *Comput Ind Eng*, vol. 97, pp. 26–40, 2016, doi: https://doi.org/10.1016/j.cie.2016.03.024.
[134] J. Schaffer, R. Caruana, L. Eshelman, and R. Das, *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization.* 1989.
[135] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, *A Racing Algorithm for Configuring Metaheuristics.* 2002.
[136] M. Birattari, *Tuning Metaheuristics*, vol. 197. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-00483-4.

[137] C. Huang, Y. Li, and X. Yao, “A Survey of Automatic Parameter Tuning Methods for Metaheuristics,” *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 201–216, 2020, doi: 10.1109/TEVC.2019.2921598.
[138] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, “F-Race and Iterated F-Race: An Overview,” in *Experimental Methods for the Analysis of Optimization Algorithms*, T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 311–336. doi: 10.1007/978-3-642-02538-9_13.
[139] D. B. Gümüş, E. Özcan, J. Atkin, and J. H. Drake, “An investigation of F-Race training strategies for cross domain optimisation with memetic algorithms,” *Inf Sci (N Y)*, vol. 619, pp. 153–171, 2023, doi: https://doi.org/10.1016/j.ins.2022.11.008.
[140] T. Stützle and M. López-Ibáñez, “Automated Design of Metaheuristic Algorithms,” in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds., Cham: Springer International Publishing, 2019, pp. 541–579. doi: 10.1007/978-3-319-91086-4_17.
[141] J. E. Smith, “Self-Adaptation in Evolutionary Algorithms for Combinatorial Optimisation,” in *Adaptive and Multilevel Metaheuristics*, C. Cotta, M. Sevaux, and K. Sörensen, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 31–57. doi: 10.1007/978-3-540-79438-7_2.
[142] L. Calvet, J. A. A, S. Carles, and R. Jana, “A statistical learning based approach for parameter fine-tuning of metaheuristics,” *SORT-Statistics and Operations Research Transactions*, vol. 40, no. 1, pp. 201–224, Jan. 2016, [Online]. Available: https://raco.cat/index.php/SORT/article/view/310078
[143] Y. Hamadi, E. Monfroy, and F. Saubion, “An Introduction to Autonomous Search,” in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–11. doi: 10.1007/978-3-642-21434-9_1.
[144] A. E. Eiben, R. Hinterding, and Z. Michalewicz, “Parameter control in evolutionary algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 124–141, 1999, doi: 10.1109/4235.771166.
[145] M. Gomes Pereira de Lacerda, L. F. de Araujo Pessoa, F. Buarque de Lima Neto, T. B. Ludermir, and H. Kuchen, “A systematic literature review on general parameter control for evolutionary and swarm-based algorithms,” *Swarm Evol Comput*, vol. 60, p. 100777, 2021, doi: https://doi.org/10.1016/j.swevo.2020.100777.
[146] M. Antoniou, R. Hribar, and G. Papa, “Parameter Control in Evolutionary Optimisation,” in *Optimization Under Uncertainty with Applications to Aerospace Engineering*, M. Vasile, Ed., Cham: Springer International Publishing, 2021, pp. 357–385. doi: 10.1007/978-3-030-60166-9_11.
[147] J. C. Costa, R. Tavares, and A. Rosa, “An experimental study on dynamic random variation of population size,” in *IEEE SMC’99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, 1999, pp. 607–612 vol.1. doi: 10.1109/ICSMC.1999.814161.
[148] R. Tanabe and A. Fukunaga, “Success-history based parameter adaptation for Differential Evolution,” in *2013 IEEE Congress on Evolutionary Computation*, 2013, pp. 71–78. doi: 10.1109/CEC.2013.6557555.
[149] D.-C. Dang and P. K. Lehre, “Self-adaptation of Mutation Rates in Non-elitist Populations,” *CoRR*, vol. abs/1606.05551, 2016, [Online]. Available: http://arxiv.org/abs/1606.05551
[150] E.-G. Talbi, “Machine Learning into Metaheuristics: A Survey and Taxonomy,” *ACM Comput. Surv.*, vol. 54, no. 6, Jul. 2021, doi: 10.1145/3459664.
[151] A. E. Eiben, M. Horvath, W. Kowalczyk, and M. C. Schut, “Reinforcement Learning for Online Control of Evolutionary Algorithms,” in *Engineering Self-Organising Systems*, S.

A. Brueckner, S. Hassas, M. Jelasity, and D. Yamins, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 151–160.

[152] J. Zhang, H. S.-H. Chung, and W.-L. Lo, "Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 3, pp. 326–335, 2007, doi: 10.1109/TEVC.2006.880727.

[153] D. Weyland, "Simulated annealing, its parameter settings and the longest common subsequence problem," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, in GECCO '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 803–810. doi: 10.1145/1389095.1389253.

[154] A. Franzin and T. Stützle, "Revisiting simulated annealing: A component-based analysis," *Comput Oper Res*, vol. 104, pp. 191–206, 2019, doi: https://doi.org/10.1016/j.cor.2018.12.015.

[155] D. Huri and T. Mankovits, "Surrogate Model-Based Parameter Tuning of Simulated Annealing Algorithm for the Shape Optimization of Automotive Rubber Bumpers," *Applied Sciences*, vol. 12, no. 11, 2022, doi: 10.3390/app12115451.

[156] R. Zeng and Y. Wang, "A chaotic simulated annealing and particle swarm improved artificial immune algorithm for flexible job shop scheduling problem," *EURASIP J Wirel Commun Netw*, vol. 2018, no. 1, p. 101, 2018, doi: 10.1186/s13638-018-1109-2.

[157] E. H. L. Aarts and P. J. M. van Laarhoven, "Statistical Cooling: A General Approach To Combinatorial Optimization Problems," *Philips Journal of Research*, vol. 40, pp. 193–226, 1985.

[158] R. E. Burkard and F. Rendl, "A thermodynamically motivated simulation procedure for combinatorial optimization problems," *Eur J Oper Res*, vol. 17, no. 2, pp. 169–174, 1984, doi: https://doi.org/10.1016/0377-2217(84)90231-5.

[159] K. Y. Tam, "A simulated annealing algorithm for allocating space to manufacturing cells," *Int J Prod Res*, vol. 30, no. 1, pp. 63–87, Jan. 1992, doi: 10.1080/00207549208942878.

[160] G. Velleda Gonzales, E. Domingues dos Santos, L. Ramos Emmendorfer, L. André Isoldi, L. A. Oliveira Rocha, and E. da Silva Diaz Estrada, "A Comparative Study of Simulated Annealing with different Cooling Schedules for Geometric Optimization of a Heat Transfer Problem According to Constructal Design," *Scientia Plena*, vol. 11, no. 8, Aug. 2015, doi: 10.14808/sci.plena.2015.081321.

[161] A. K. Peprah, S. K. Appiah, and S. K. Amponsah, "An Optimal Cooling Schedule Using a Simulated Annealing Based Approach," *Appl Math (Irvine)*, vol. 8, pp. 1195–1210, 2017, doi: 10.4236/am.2017.88090.

[162] Yaghout Nourani and Bjarne Andresen, "A comparison of simulated annealing cooling strategies," *J Phys A Math Gen*, vol. 31, no. 41, p. 8373, 1998, doi: 10.1088/0305-4470/31/41/011.

[163] H.-S. Cho, C.-H. Paik, H.-M. Yoon, and H.-G. Kim, "A robust design of simulated annealing approach for mixed-model sequencing," *Comput Ind Eng*, vol. 48, no. 4, pp. 753–764, 2005, doi: https://doi.org/10.1016/j.cie.2004.12.005.

[164] J. and M. W. Aarts Emile and Korst, "Simulated Annealing," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, G. Burke Edmund K. and Kendall, Ed., Boston, MA: Springer US, 2005, pp. 187–210. doi: 10.1007/0-387-28356-0_7.

[165] J. Rose, W. Klebsch, and J. Wolf, "Temperature measurement and equilibrium dynamics of simulated annealing placements," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 3, pp. 253–259, 1990, doi: 10.1109/43.46801.

[166] M. S. Hussin and T. Stützle, “Tabu search vs. simulated annealing for solving large quadratic assignment instances,” *Comput Oper Res*, vol. 43, pp. 286–291, 2014.
[167] D. Abramson, “Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms,” *Manage Sci*, vol. 37, no. 1, pp. 98–113, 1991, [Online]. Available: http://www.jstor.org/stable/2632495
[168] I. H. Osman and C. N. Potts, “Simulated annealing for permutation flow-shop scheduling,” *Omega (Westport)*, vol. 17, no. 6, pp. 551–557, 1989, doi: https://doi.org/10.1016/0305-0483(89)90059-5.
[169] Y. Crama and M. Schyns, “Simulated annealing for complex portfolio selection problems,” *Eur J Oper Res*, vol. 150, no. 3, pp. 546–571, 2003, doi: https://doi.org/10.1016/S0377-2217(02)00784-1.
[170] P. Siarry, G. Berthiau, F. Durdin, and J. Haussy, “Enhanced simulated annealing for globally minimizing functions of many-continuous variables,” *ACM Trans. Math. Softw.*, vol. 23, no. 2, pp. 209–228, Jun. 1997, doi: 10.1145/264029.264043.
[171] S. Raschka, J. Patterson, and C. Nolet, “Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence,” 2020. doi: 10.3390/info11040193.
[172] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. London, England: The MIT Press, 2012.
[173] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull Math Biophys*, vol. 5, no. 4, pp. 115–133, 1943, doi: 10.1007/BF02478259.
[174] A. M. Turing, “Computing Machinery and Intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, Oct. 1950.
[175] K. El Bouchefry and R. S. de Souza, “Learning in Big Data: Introduction to Machine Learning,” in *Knowledge Discovery in Big Data from Astronomy and Earth Observation: Astrogeoinformatics*, 2020, pp. 225–249. doi: 10.1016/B978-0-12-819154-5.00023-0.
[176] A. L. Samuel, “Some Studies in Machine Learning Using The Game of Checkers,” *IBM J Res Dev*, vol. 3, no. 3, pp. 210–229, 1959.
[177] T. M. (Tom M. Mitchell, *Machine Learning*, vol. 4. New York, NY, USA: McGraw-Hill, Inc., 1997.
[178] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” 2015. doi: 10.1126/science.aaa8415.
[179] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, 2nd ed., no. 1. London, England: MIT Press, 2018.
[180] A. V Joshi, “Essential Concepts in Artificial Intelligence and Machine Learning,” in *Machine Learning and Artificial Intelligence*, A. V Joshi, Ed., Cham: Springer International Publishing, 2020, pp. 9–20. doi: 10.1007/978-3-030-26622-6_2.
[181] S. Sah, “Machine Learning: A Review of Learning Types,” *Preprints (Basel)*, Jul. 2020, doi: 10.20944/preprints202007.0230.v1.
[182] C. Janiesch, P. Zschech, and K. Heinrich, “Machine learning and deep learning,” *Electronic Markets*, vol. 31, no. 3, 2021, doi: 10.1007/s12525-021-00475-2.
[183] T. Jiang, J. L. Gradus, and A. J. Rosellini, “Supervised Machine Learning: A Brief Primer,” *Behav Ther*, vol. 51, no. 5, pp. 675–687, 2020, doi: 10.1016/j.beth.2020.05.002.
[184] J. Ha, M. Kambe, and J. Pe, *Data Mining: Concepts and Techniques*. 2011. doi: 10.1016/C2009-0-61819-5.
[185] Aurélien Géron, *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow*, 3rd ed. Sebastopol, CA: O’Reilly Media, Inc., 2022.
[186] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. New York, NY: Springer, 2006.

[187] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement Learning: A Survey,” *J. Artif. Int. Res.*, vol. 4, no. 1, pp. 237–285, May 1996.
[188] I. H. Sarker, “Machine Learning: Algorithms, Real-World Applications and Research Directions,” 2021. doi: 10.1007/s42979-021-00592-x.
[189] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.
[190] C. Szepesvári, *Algorithms for Reinforcement Learning*, 1st ed. Morgan & Claypool Publishers, 2010.
[191] R. Nian, J. Liu, and B. Huang, “A review On reinforcement learning: Introduction and applications in industrial process control,” *Comput Chem Eng*, vol. 139, p. 106886, 2020, doi: https://doi.org/10.1016/j.compchemeng.2020.106886.
[192] B. Cunha, A. Madureira, B. Fonseca, and J. Matos, “Intelligent Scheduling with Reinforcement Learning,” *Applied Sciences*, vol. 11, no. 8, p. 3710, 2021, doi: 10.3390/app11083710.
[193] B. Cunha, A. Madureira, and B. Fonseca, “A Reinforcement Learning Environment for Job Shop Scheduling Problems,” *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 12, pp. 231–238, 2020, [Online]. Available: www.mirlabs.net/ijcisim/index.html
[194] C. Yu, J. Liu, S. Nemati, and G. Yin, “Reinforcement Learning in Healthcare: A Survey,” *ACM Comput Surv*, vol. 55, no. 1, 2021, doi: 10.1145/3477600.
[195] J. R. Vázquez-Canteli and Z. Nagy, “Reinforcement learning for demand response: A review of algorithms and modeling techniques,” *Appl Energy*, vol. 235, pp. 1072–1089, 2019, doi: https://doi.org/10.1016/j.apenergy.2018.11.002.
[196] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-Learning,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, in AAAI’16. AAAI Press, 2016, pp. 2094–2100.
[197] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *Int J Rob Res*, vol. 32, no. 11, pp. 1238–1274, Aug. 2013, doi: 10.1177/0278364913495721.
[198] B. Cunha, A. M. Madureira, B. Fonseca, and D. Coelho, “Deep Reinforcement Learning as a Job Shop Scheduling Solver: A Literature Review,” in *Hybrid Intelligent Systems*, A. M. Madureira, A. Abraham, N. Gandhi, and M. L. Varela, Eds., Cham: Springer International Publishing, 2020, pp. 350–359.
[199] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust Region Policy Optimization,” in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., in Proceedings of Machine Learning Research, vol. 37. Lille, France: PMLR, Mar. 2015, pp. 1889–1897. [Online]. Available: https://proceedings.mlr.press/v37/schulman15.html
[200] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, in ICML’14. JMLR.org, 2014, pp. I–387–I–395.
[201] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep Reinforcement Learning: A Brief Survey,” *IEEE Signal Process Mag*, vol. 34, no. 6, pp. 26–38, 2017, doi: 10.1109/MSP.2017.2743240.
[202] B. Cunha, A. Madureira, and B. Fonseca, “Reinforcement Learning Environment for Job Shop Scheduling Problems,” *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 12, pp. 231–238, 2020, [Online]. Available: www.mirlabs.net/ijcisim/index.html
[203] M. G. Bellemare, W. Dabney, and R. Munos, “A Distributional Perspective on Reinforcement Learning,” in *Proceedings of the 34th International Conference on*

Machine Learning, D. Precup and Y. W. Teh, Eds., in Proceedings of Machine Learning Research, vol. 70. PMLR, Mar. 2017, pp. 449–458. [Online]. Available: https://proceedings.mlr.press/v70/bellemare17a.html

[204] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous Deep Q-Learning with Model-based Acceleration," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., in Proceedings of Machine Learning Research, vol. 48. New York, New York, USA: PMLR, Mar. 2016, pp. 2829–2838. [Online]. Available: https://proceedings.mlr.press/v48/gu16.html

[205] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and Autonomous Control Using Reinforcement Learning: A Survey," *IEEE Trans Neural Netw Learn Syst*, vol. 29, no. 6, pp. 2042–2062, 2018, doi: 10.1109/TNNLS.2017.2773458.

[206] F. AlMahamid and K. Grolinger, "Reinforcement Learning Algorithms: An Overview and Classification," *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–7, 2021, [Online]. Available: https://api.semanticscholar.org/CorpusID:240003074

[207] J. Clifton and E. Laber, "Q-Learning: Theory and Applications," *Annu Rev Stat Appl*, vol. 7, no. 1, pp. 279–301, Mar. 2020, doi: 10.1146/annurev-statistics-031219-041220.

[208] C. Watkins, "Learning From Delayed Rewards," Ph.D Thesis, University of Cambridge, England, 1989.

[209] C. J. C. H. Watkins and P. Dayan, "Technical Note: Q-Learning," *Mach Learn*, vol. 8, no. 3, pp. 279–292, 1992, doi: 10.1023/A:1022676722315.

[210] Q. Huang, "Model-Based or Model-Free, a Review of Approaches in Reinforcement Learning," in *2020 International Conference on Computing and Data Science (CDS)*, 2020, pp. 219–221. doi: 10.1109/CDS49703.2020.00051.

[211] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Comput Oper Res*, vol. 134, p. 105400, 2021, doi: https://doi.org/10.1016/j.cor.2021.105400.

[212] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-Learning Algorithms: A Comprehensive Classification and Applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

[213] E. Even-Dar and Y. Mansour, "Learning Rates for Q-learning," *J. Mach. Learn. Res.*, vol. 5, pp. 1–25, Dec. 2004.

[214] M. Karimi-Mamaghan, M. Mohammadi, B. Pasdeloup, and P. Meyer, "Learning to select operators in meta-heuristics: An integration of Q-learning into the iterated greedy algorithm for the permutation flowshop scheduling problem," *Eur J Oper Res*, vol. 304, no. 3, pp. 1296–1330, 2023, doi: https://doi.org/10.1016/j.ejor.2022.03.054.

[215] D. Qiao, L. Duan, H. Li, and Y. Xiao, "Optimization of job shop scheduling problem based on deep reinforcement learning," *Evol Intell*, vol. 17, no. 1, pp. 371–383, 2024, doi: 10.1007/s12065-023-00885-5.

[216] E. S. Low, P. Ong, and K. C. Cheah, "Solving the optimal path planning of a mobile robot using improved Q-learning," *Rob Auton Syst*, vol. 115, pp. 143–161, 2019, doi: https://doi.org/10.1016/j.robot.2019.02.013.

[217] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-Learning Algorithms: A Comprehensive Classification and Applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

[218] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *J. Artif. Int. Res.*, vol. 4, no. 1, pp. 237–285, May 1996.

[219] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi, "Machine learning at the service of meta-heuristics for solving combinatorial

optimization problems: A state-of-the-art," *Eur J Oper Res*, vol. 296, no. 2, pp. 393–422, 2022, doi: https://doi.org/10.1016/j.ejor.2021.04.032.

[220] J. R. Rice, "The Algorithm Selection Problem," in *Advances in Computers*, vol. 15, M. Rubinoff and M. C. Yovits, Eds., Elsevier, 1976, pp. 65–118. doi: https://doi.org/10.1016/S0065-2458(08)60520-3.

[221] L. Kotthoff, "Algorithm Selection for Combinatorial Search Problems: A Survey," in *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, C. Bessiere, L. De Raedt, L. Kotthoff, S. Nijssen, B. O'Sullivan, and D. Pedreschi, Eds., Cham: Springer International Publishing, 2016, pp. 149–190. doi: 10.1007/978-3-319-50137-6_7.

[222] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: portfolio-based algorithm selection for SAT," *J. Artif. Int. Res.*, vol. 32, no. 1, pp. 565–606, Jun. 2008.

[223] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A Classification of Hyper-Heuristic Approaches: Revisited," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds., Cham: Springer International Publishing, 2019, pp. 453–477. doi: 10.1007/978-3-319-91086-4_14.

[224] E. and S. Y. Leyton-Brown Kevin and Nudelman, "Learning the Empirical Hardness of Optimization Problems: The Case of Combinatorial Auctions," in *Principles and Practice of Constraint Programming - CP 2002*, P. Van Hentenryck, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 556–572.

[225] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Automatic Design of a Hyper-Heuristic Framework With Gene Expression Programming for Combinatorial Optimization Problems," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 309–325, 2015, doi: 10.1109/TEVC.2014.2319051.

[226] Q. Li, S.-Y. Liu, and X.-S. Yang, "Influence of initialization on the performance of metaheuristic optimizers," *Appl Soft Comput*, vol. 91, p. 106193, 2020, doi: https://doi.org/10.1016/j.asoc.2020.106193.

[227] M. Sarhani, S. Voß, and R. Jovanovic, "Initialization of metaheuristics: comprehensive review, critical analysis, and research directions," *International Transactions in Operational Research*, vol. 30, no. 6, pp. 3361–3397, Nov. 2023, doi: https://doi.org/10.1111/itor.13237.

[228] B. Kazimipour, X. Li, and A. K. Qin, "A review of population initialization techniques for evolutionary algorithms," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014, pp. 2585–2592. doi: 10.1109/CEC.2014.6900618.

[229] P. Victer Paul, N. Moganarangan, S. S. Kumar, R. Raju, T. Vengattaraman, and P. Dhavachelvan, "Performance analyses over population seeding techniques of the permutation-coded genetic algorithm: An empirical study based on traveling salesman problems," *Appl Soft Comput*, vol. 32, pp. 383–402, 2015, doi: https://doi.org/10.1016/j.asoc.2015.03.038.

[230] D. Catteeuw, M. Drugan, and B. Manderick, " Guided Restarts Hill-Climbing," in *Conference5th International Conference on Metaheuristics and Nature Inspired Computing, META'14*, 2014.

[231] M. Vermorel Joannès and Mohri, "Multi-armed Bandit Algorithms and Empirical Evaluation," in *Machine Learning: ECML 2005*, R. and B. P. B. and J. A. M. and T. L. Gama João and Camacho, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 437–448.

[232] A. Tuson, "Adapting Operator Probabilities In Genetic Algorithms," Jul. 1998.

[233] J. Pei, J. Liu, and Y. Mei, "Learning from Offline and Online Experiences: A Hybrid Adaptive Operator Selection Framework," in *Proceedings of the Genetic and*

Evolutionary Computation Conference, in GECCO '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1017–1025. doi: 10.1145/3638529.3654062.

[234] R. Durgut, M. E. Aydin, and A. Rakib, "Transfer Learning for Operator Selection: A Reinforcement Learning Approach," *Algorithms*, vol. 15, no. 1, 2022, doi: 10.3390/a15010024.

[235] S. D. Handoko, D. T. Nguyen, Z. Yuan, and H. C. Lau, "Reinforcement learning for adaptive operator selection in memetic search applied to quadratic assignment problem," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, in GECCO Comp '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 193–194. doi: 10.1145/2598394.2598451.

[236] J. Pei, Y. Mei, J. Liu, and X. Yao, "An Investigation of Adaptive Operator Selection in Solving Complex Vehicle Routing Problem," in *PRICAI 2022: Trends in Artificial Intelligence: 19th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2022, Shanghai, China, November 10–13, 2022, Proceedings, Part I*, Berlin, Heidelberg: Springer-Verlag, 2022, pp. 562–573. doi: 10.1007/978-3-031-20862-1_41.

[237] M. E. Aydin, R. Durgut, and A. Rakib, "Adaptive operator selection utilising generalised experience," *ArXiv*, vol. abs/2401.05350, 2023, [Online]. Available: https://api.semanticscholar.org/CorpusID:266933147

[238] K. Li, Á. Fialho, S. Kwong, and Q. Zhang, "Adaptive Operator Selection With Bandits for a Multiobjective Evolutionary Algorithm Based on Decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 1, pp. 114–130, 2014, doi: 10.1109/TEVC.2013.2239648.

[239] Y. Tian, X. Li, H. Ma, X. Zhang, K. C. Tan, and Y. Jin, "Deep Reinforcement Learning Based Adaptive Operator Selection for Evolutionary Multi-Objective Optimization," *IEEE Trans Emerg Top Comput Intell*, vol. 7, no. 4, pp. 1051–1064, 2023, doi: 10.1109/TETCI.2022.3146882.

[240] W. Yi, R. Qu, L. Jiao, and B. Niu, "Automated Design of Metaheuristics Using Reinforcement Learning Within a Novel General Search Framework," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 4, pp. 1072–1084, 2023, doi: 10.1109/TEVC.2022.3197298.

[241] M. Sharma, A. Komninos, M. López-Ibáñez, and D. Kazakov, "Deep reinforcement learning based parameter control in differential evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, in GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 709–717. doi: 10.1145/3321707.3321813.

[242] W. Yi, R. Qu, L. Jiao, and B. Niu, "Automated Design of Metaheuristics Using Reinforcement Learning Within a Novel General Search Framework," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 4, pp. 1072–1084, 2023, doi: 10.1109/TEVC.2022.3197298.

[243] H. Lu, X. Zhang, and S. Yang, "A Learning-based Iterative Method for Solving Vehicle Routing Problems," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:213389645

[244] H. Chung and J. Alonso, "Mutiobjective Optimization Using Approximation Model-Based Genetic Algorithms," Aug. 2004, doi: 10.2514/6.2004-4325.

[245] D. Chafekar, L. Shi, K. Rasheed, and J. Xuan, "Multiobjective GA Optimization Using Reduced Models," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 35, pp. 261–265, Jun. 2005, doi: 10.1109/TSMCC.2004.841905.

[246] Y. Jin, “A comprehensive survey of fitness approximation in evolutionary computation,” *Soft comput*, vol. 9, no. 1, pp. 3–12, 2005, doi: 10.1007/s00500-003-0328-5.
[247] J.-W. Yoon and S.-B. Cho, “Fitness approximation for genetic algorithm using combination of approximation model and fuzzy clustering technique,” in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–6. doi: 10.1109/CEC.2010.5586519.
[248] I. Tzruia, T. Halperin, M. Sipper, and A. Elyasaf, *Fitness Approximation through Machine Learning*. 2023.
[249] K. Shi L. and Rasheed, “A Survey of Fitness Approximation Methods Applied in Evolutionary Algorithms,” in *Computational Intelligence in Expensive Optimization Problems*, C.-K. Tenne Yoel and Goh, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 3–28. doi: 10.1007/978-3-642-10701-6_1.
[250] I. R. Mekki, A. Cherrered, F. B.-S. Tayeb, and K. Benatchba, “Fitness Approximation Surrogate-assisted Hyper-heuristic for the Permutation Flowshop Problem,” *Procedia Comput Sci*, vol. 225, pp. 4043–4054, 2023, doi: https://doi.org/10.1016/j.procs.2023.10.400.
[251] T. Bartz-Beielstein and M. Zaefferer, “Model-based methods for continuous and discrete global optimization,” *Appl Soft Comput*, vol. 55, pp. 154–167, 2017, doi: https://doi.org/10.1016/j.asoc.2017.01.039.
[252] S.-C. Horng, S.-Y. Lin, L. H. Lee, and C.-H. Chen, “Memetic Algorithm for Real-Time Combinatorial Stochastic Simulation Optimization Problems With Performance Analysis,” *IEEE Trans Cybern*, vol. 43, no. 5, pp. 1495–1509, 2013, doi: 10.1109/TCYB.2013.2264670.
[253] A. Rosales-Pérez, C. A. C. Coello, J. A. Gonzalez, C. A. Reyes-Garcia, and H. J. Escalante, “A hybrid surrogate-based approach for evolutionary multi-objective optimization,” in *2013 IEEE Congress on Evolutionary Computation*, 2013, pp. 2548–2555. doi: 10.1109/CEC.2013.6557876.
[254] M. Reyes-Sierra and C. A. C. Coello, “Fitness inheritance in multi-objective particle swarm optimization,” in *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, 2005, pp. 116–123. doi: 10.1109/SIS.2005.1501610.
[255] R. E. Smith, B. A. Dike, and S. A. Stegmann, “Fitness inheritance in genetic algorithms,” in *Proceedings of the 1995 ACM Symposium on Applied Computing*, in SAC ’95. New York, NY, USA: Association for Computing Machinery, 1995, pp. 345–350. doi: 10.1145/315891.316014.
[256] H.-S. Kim and S.-B. Cho, “An efficient genetic algorithm with less fitness evaluation by clustering,” in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, 2001, pp. 887–894 vol. 2. doi: 10.1109/CEC.2001.934284.
[257] N. Donthu, S. Kumar, D. Mukherjee, N. Pandey, and W. M. Lim, “How to conduct a bibliometric analysis: An overview and guidelines,” *J Bus Res*, vol. 133, pp. 285–296, 2021, doi: https://doi.org/10.1016/j.jbusres.2021.04.070.
[258] R. Briner and D. Denyer, “Systematic Review and Evidence Synthesis as a Practice and Scholarship Tool,” in *Handbook of evidence-based management: Companies, classrooms and research*, 2012, pp. 112–129. doi: 10.1093/oxfordhb/9780199763986.013.0007.
[259] N. J. van Eck and L. Waltman, “Software survey: VOSviewer, a computer program for bibliometric mapping,” *Scientometrics*, vol. 84, no. 2, pp. 523–538, 2010, doi: 10.1007/s11192-009-0146-3.
[260] V. A. Tatsis and K. E. Parsopoulos, “Reinforced Online Parameter Adaptation Method for Population-based Metaheuristics,” in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 360–367. doi: 10.1109/SSCI47803.2020.9308488.

[261] S. Lessmann, M. Caserta, and I. M. Arango, “Tuning metaheuristics: A data mining based approach for particle swarm optimization,” *Expert Syst Appl*, vol. 38, no. 10, pp. 12826–12838, 2011, doi: https://doi.org/10.1016/j.eswa.2011.04.075.
[262] T. N. Huynh, D. T. T. Do, and J. Lee, “Q-Learning-based parameter control in differential evolution for structural optimization,” *Appl Soft Comput*, vol. 107, p. 107464, 2021, doi: https://doi.org/10.1016/j.asoc.2021.107464.
[263] A. A. Chaves and L. H. N. Lorena, “An Adaptive and Near Parameter-Free BRKGA Using Q-Learning Method,” in *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 2331–2338. doi: 10.1109/CEC45853.2021.9504766.
[264] M. Pikalov and A. Pismerov, “Exploratory Landscape Analysis Based Parameter Control,” in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, in GECCO ’23 Companion. New York, NY, USA: Association for Computing Machinery, 2023, pp. 2378–2381. doi: 10.1145/3583133.3596364.
[265] A. Smirnov and V. Mironovich, “Towards landscape-aware parameter tuning for the (1 + (λ, λ)) genetic algorithm for permutations,” in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, in GECCO ’22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 2326–2329. doi: 10.1145/3520304.3534007.
[266] R. Reijnen, Y. Zhang, Z. Bukhsh, and M. Guzek, “Learning to Adapt Genetic Algorithms for Multi-Objective Flexible Job Shop Scheduling Problems,” in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, in GECCO ’23 Companion. New York, NY, USA: Association for Computing Machinery, 2023, pp. 315–318. doi: 10.1145/3583133.3590700.
[267] N. Caselli, R. Soto, B. Crawford, S. Valdivia, and R. Olivares, “A Self-Adaptive Cuckoo Search Algorithm Using a Machine Learning Technique,” *Mathematics*, vol. 9, no. 16, 2021, doi: 10.3390/math9161840.
[268] A. Aleti, I. Moser, I. Meedeniya, and L. Grunske, “Choosing the Appropriate Forecasting Model for Predictive Parameter Control,” *Evol Comput*, vol. 22, Jul. 2013, doi: 10.1162/EVCO_a_00113.
[269] J. Sun, X. Liu, T. Bäck, and Z. Xu, “Learning Adaptive Differential Evolution Algorithm From Optimization Experiences by Policy Gradient,” *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 666–680, 2021, doi: 10.1109/TEVC.2021.3060811.
[270] G. Karafotias, A. E. Eiben, and M. Hoogendoorn, “Generic parameter control with reinforcement learning,” in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, in GECCO ’14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 1319–1326. doi: 10.1145/2576768.2598360.
[271] M. G. P. de Lacerda, F. B. de Lima Neto, T. B. Ludermir, and H. Kuchen, “Out-of-the-box parameter control for evolutionary and swarm-based algorithms with distributed reinforcement learning,” *Swarm Intelligence*, vol. 17, no. 3, pp. 173–217, 2023, doi: 10.1007/s11721-022-00222-z.
[272] A. de Miguel Gomez and F. Toosi, “Continuous Parameter Control in Genetic Algorithms using Policy Gradient Reinforcement Learning,” in *Proceedings of the 13th International Joint Conference on Computational Intelligence*, SCITEPRESS - Science and Technology Publications, 2021, pp. 115–122. doi: 10.5220/0010643500003063.
[273] T. C. Bora, L. Lebensztajn, and L. D. S. Coelho, “Non-Dominated Sorting Genetic Algorithm Based on Reinforcement Learning to Optimization of Broad-Band Reflector Antennas Satellite,” *IEEE Trans Magn*, vol. 48, no. 2, pp. 767–770, 2012, doi: 10.1109/TMAG.2011.2177076.

[274] R. Pavón, F. Díaz, R. Laza, and V. Luzón, “Automatic parameter tuning with a Bayesian case-based reasoning system. A case of study,” *Expert Syst Appl*, vol. 36, no. 2, Part 2, pp. 3407–3420, 2009, doi: https://doi.org/10.1016/j.eswa.2008.02.044.
[275] J. Silc, K. Taškova, and P. Korošec, “Data mining-assisted parameter tuning of a search algorithm,” *Informatica (Slovenia)*, vol. 39, pp. 169–176, Jul. 2015.
[276] I. Pereira and A. Madureira, “Racing based approach for Metaheuristics parameter tuning,” in *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*, 2015, pp. 1–6. doi: 10.1109/CISTI.2015.7170351.
[277] J. Tondut, C. Ollier, N. Di Cesare, J. C. Roux, and S. Ronel, “An automatic kriging machine learning method to calibrate meta-heuristic algorithms for solving optimization problems,” *Eng Appl Artif Intell*, vol. 113, p. 104940, 2022, doi: https://doi.org/10.1016/j.engappai.2022.104940.
[278] H. Xiong, S. Shi, D. Ren, and J. Hu, “A survey of job shop scheduling problem: The types and models,” *Comput Oper Res*, vol. 142, p. 105731, 2022, doi: https://doi.org/10.1016/j.cor.2022.105731.
[279] T. C. E. Cheng, B. Peng, and Z. Lü, “A hybrid evolutionary algorithm to solve the job shop scheduling problem,” *Ann Oper Res*, vol. 242, no. 2, pp. 223–237, 2016, doi: 10.1007/s10479-013-1332-5.
[280] Z. Li, C. Zhao, G. Zhang, D. Zhu, and L. Cui, “Multi-strategy improved sparrow search algorithm for job shop scheduling problem,” *Cluster Comput*, 2023, doi: 10.1007/s10586-023-04200-w.
[281] E. Guzman, B. Andres, and R. Poler, “Matheuristic Algorithm for Job-Shop Scheduling Problem Using a Disjunctive Mathematical Model,” *Computers*, vol. 11, no. 1, 2022, doi: 10.3390/computers11010001.
[282] J. Błażewicz, E. Pesch, and M. Sterna, “The disjunctive graph machine representation of the job shop scheduling problem,” *Eur J Oper Res*, vol. 127, no. 2, pp. 317–331, 2000, doi: https://doi.org/10.1016/S0377-2217(99)00486-5.
[283] R. Qing-dao-er-ji and Y. Wang, “A new hybrid genetic algorithm for job shop scheduling problem,” *Comput Oper Res*, vol. 39, no. 10, pp. 2291–2299, 2012, doi: https://doi.org/10.1016/j.cor.2011.12.005.
[284] K. Akram, K. Kamal, and A. Zeb, “Fast simulated annealing hybridized with quenching for solving job shop scheduling problem,” *Appl Soft Comput*, vol. 49, pp. 510–523, 2016, doi: https://doi.org/10.1016/j.asoc.2016.08.037.
[285] S. Lawrence, “Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement),” Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.
[286] J. Adams, E. Balas, and D. Zawack, “The Shifting Bottleneck Procedure for Job Shop Scheduling,” *Manage Sci*, vol. 34, no. 3, pp. 391–401, 1988, [Online]. Available: http://www.jstor.org/stable/2632051
[287] E. Taillard, “Benchmarks for basic scheduling problems,” *Eur J Oper Res*, vol. 64, no. 2, pp. 278–285, 1993, doi: https://doi.org/10.1016/0377-2217(93)90182-M.
[288] M. Gendreau and J.-Y. Potvin, “Handbook of Metaheuristics,” in *International Series in Operations Research & Management Science Volume 272*, 3rd ed., 2019.
[289] S. Zhan, J. Lin, Z. Zhang, and Y. Zhong, “List-Based Simulated Annealing Algorithm for Traveling Salesman Problem,” *Comput Intell Neurosci*, vol. 2016, p. 1712630, 2016, doi: 10.1155/2016/1712630.
[290] F. Romeo and A. Sangiovanni-Vincentelli, “Probabilistic Hill Climbing Algorithms: Properties and Applications,” in *Proceedings of the 1985 Chapel Hill Conference on VLSI*, 1985, pp. 393–417.

[291] M. A. Wiering and H. van Hasselt, “The QV family compared to other reinforcement learning algorithms,” in *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2009, pp. 101–108. doi: 10.1109/ADPRL.2009.4927532.
[292] D. Dias, A. S. Santos, and L. R. Varela, “Machine Learning Algorithms in Scheduling Problems: An Overview and Future Paths,” in *Innovations in Mechatronics Engineering III*, J. Machado, F. Soares, J. Trojanowska, S. Yildirim, J. Vojtěšek, P. Rea, B. Gramescu, and O. O. Hrybiuk, Eds., Cham: Springer Nature Switzerland, 2024, pp. 79–89.
[293] C. D. Hubbs, C. Li, N. V Sahinidis, I. E. Grossmann, and J. M. Wassick, “A deep reinforcement learning approach for chemical production scheduling,” *Comput Chem Eng*, vol. 141, p. 106982, 2020, doi: https://doi.org/10.1016/j.compchemeng.2020.106982.
[294] C.-L. Liu, C.-C. Chang, and C.-J. Tseng, “Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems,” *IEEE Access*, vol. 8, pp. 71752–71762, 2020, doi: 10.1109/ACCESS.2020.2987820.
[295] X. Wang and L. Tang, “A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem,” *Comput Oper Res*, vol. 79, pp. 60–77, 2017, doi: https://doi.org/10.1016/j.cor.2016.10.003.
[296] A. Uzunoglu, C. Gahm, and A. Tuma, “A machine learning enhanced multi-start heuristic to efficiently solve a serial-batch scheduling problem,” *Ann Oper Res*, 2023, doi: 10.1007/s10479-023-05541-w.
[297] L. Mönch, J. Zimmermann, and P. Otto, “Machine learning techniques for scheduling jobs with incompatible families and unequal ready times on parallel batch machines,” *Eng Appl Artif Intell*, vol. 19, no. 3, pp. 235–245, 2006, doi: https://doi.org/10.1016/j.engappai.2005.10.001.
[298] L. Lin and M. Gen, “Hybrid evolutionary optimisation with learning for production scheduling: state-of-the-art survey on algorithms and applications,” *Int J Prod Res*, vol. 56, no. 1–2, pp. 193–223, Jan. 2018, doi: 10.1080/00207543.2018.1437288.
[299] H. Togo, K. Asanuma, T. Nishi, and Z. Liu, “Machine Learning and Inverse Optimization for Estimation of Weighting Factors in Multi-Objective Production Scheduling Problems,” *Applied Sciences*, vol. 12, no. 19, p. 9472, 2022, doi: 10.3390/app12199472.
[300] M. H. Fazel Zarandi, A. A. Sadat Asl, S. Sotudian, and O. Castillo, “A state of the art review of intelligent scheduling,” *Artif Intell Rev*, vol. 53, no. 1, pp. 501–593, 2020, doi: 10.1007/s10462-018-9667-6.

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade. Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Declaro que o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

NOME: Daniel Joaquim da Silva Dias

ISEP, Porto, 13 de setembro de 2024

Apêndice A

Grid Word Problem

```
import numpy as np
import operator
import matplotlib.pyplot as plt

class GridWorld:
    def __init__(self):
        self.height = 5
        self.width = 5
        self.grid = np.zeros(( self.height, self.width)) - 1
        self.bomb_location = (1,3)
        self.gold_location = (0,3)
        self.terminal_states = [ self.bomb_location, self.gold_location]
        self.grid[ self.bomb_location[0], self.bomb_location[1]] = -10
        self.grid[ self.gold_location[0], self.gold_location[1]] = 10
        self.actions = ['UP', 'DOWN', 'LEFT', 'RIGHT']

    def available_actions(self):
        return self.actions

    def reward(self, next_location):
        return self.grid[ next_location[0], next_location[1]]

    def make_step(self, action, state):
        current_location = state

        if action == 'UP':
            if current_location[0] == 0:
                next_location = current_location
                reward = self.reward(next_location)
            else:
                next_location = ( current_location[0] - 1,
current_location[1])
                reward = self.reward(next_location)

        elif action == 'DOWN':
```

```
            if current_location[0] == self.height - 1:
                next_location = current_location
                reward = self.reward(next_location)
            else:
                next_location = ( current_location[0] + 1,
current_location[1])
                reward = self.reward(next_location)

        elif action == 'LEFT':
            if current_location[1] == 0:
                next_location = current_location
                reward = self.reward(next_location)
            else:
                next_location = ( current_location[0],
current_location[1] - 1)
                reward = self.reward(next_location)

        elif action == 'RIGHT':
            if current_location[1] == self.width - 1:
                next_location = current_location
                reward = self.reward(next_location)
            else:
                next_location = ( current_location[0],
current_location[1] + 1)
                reward = self.reward(next_location)

        return reward, next_location

    def check_state(self, location):
        if location in self.terminal_states:
            return 'TERMINAL'

    def initial_state(self):
        current_location = ( 4, np.random.randint(0,5))
        return current_location

class RandomAgent():
    def select_action(self, available_actions):
        return np.random.choice(available_actions)

class QLearningAgent():
    def __init__(self, environment, epsilon=0.05, alpha=0.1, gamma=1):
        self.environment = environment
        self.q_table = dict()
        for x in range(environment.height):
            for y in range(environment.width):
```

```
                self.q_table[(x,y)] = {'UP':0, 'DOWN':0, 'LEFT':0,
'RIGHT':0}

        self.epsilon = epsilon
        self.alpha = alpha
        self.gamma = gamma

    def select_action(self, current_location, available_actions):
        if np.random.uniform(0,1) < self.epsilon:
            return self.explore_action(available_actions)
        else:
            return self.exploit_action(current_location)

    def explore_action(self, available_actions):
        action = np.random.choice(available_actions)
        return action

    def exploit_action(self, current_location):
        q_values_state = self.q_table[current_location]
        max_q_value = max(q_values_state.values())
        action = np.random.choice([k for k, v in q_values_state.items()
if v == max_q_value])
        return action

    def update_q_table(self, old_state, reward, new_state, action):
        current_q_value = self.q_table[old_state][action]
        max_next_q_value = max(self.q_table[new_state]. values())
        new_q_value = (1 - self.alpha) * current_q_value + self.alpha *
(reward + self.gamma * max_next_q_value)

        self.q_table[old_state][action] = new_q_value

def play(environment, agent, trials=500, max_steps_per_episode=1000,
learn=False):
    reward_per_episode = []
    for trial in range(trials):
        cumulative_reward = 0
        step = 0
        game_over = False
        state = environment.initial_state()
        while step < max_steps_per_episode and game_over != True:
            action = agent.select_action(state, environment.actions)
            reward, new_state = environment.make_step(action, state)

            if learn == True:
                agent.update_q_table(state, reward, new_state, action)
```

```
            cumulative_reward += reward
            step += 1
            state = new_state

            if environment.check_state(new_state) == 'TERMINAL':
                environment.__init__()
                game_over = True

        reward_per_episode.append(cumulative_reward)
    return reward_per_episode

# Agente Random
environment = GridWorld()
random_agent = RandomAgent()
reward_per_episode = play(environment, random_agent, trials=500)

# Agente Q-Learning
environment = GridWorld()
agentQ = QLearningAgent(environment)
reward_per_episode = play(environment, agentQ, trials=500, learn=True)

plt.plot(reward_per_episode)
plt.show()
```

Apêndice B

Análise Bibliométrica

Para perceber a relevância e o potencial impacto do trabalho proposto nesta dissertação, é fundamental entender as contribuições passadas e presentes que se encontram na literatura, e identificar padrões, lacunas e tendências emergentes. Assim, não só se posiciona o trabalho no contexto atual, como também se constrói um conhecimento aprofundado sobre potenciais aspetos que o mesmo pode explorar [257].

Uma revisão da literatura é um processo metódico que tem o objetivo de analisar o estado de arte de um determinado domínio temático [258]. O principal objetivo neste trabalho é perceber como é que os algoritmos de ML têm sido utilizados para resolver Problemas de Escalonamento. Neste caso, para a recolha de dados, isto é, obter o conjunto de publicações a analisar, utilizou-se a base de dados *Web of Science* porque é reconhecida como um dos maiores repositórios de documentos científicos, o que significa que contém uma vasta coleção de publicações de investigação.

A Tabela 16 apresenta a *query* usada, onde os termos estão em inglês, e os resultados obtidos, em termos de número de publicações relacionadas com o tema deste trabalho, da pesquisa realizada durante o mês de dezembro de 2023. Esta solução final foi construída após vários estudos, e é importante mencionar os seguintes aspetos:

- → Para referir "Escalonamento", utilizaram-se três dos termos mais comuns usados na literatura, de forma a obter uma pesquisa inclusiva. Adicionar termos como "*meta-heuristics*", iria reduzir significativamente o número de publicações e abrir a possibilidade de artigos relevantes nesta área serem omitidos, principalmente os que abordam a introdução de RL para resolver problemas de Escalonamento;
- → Para o caso do "*Machine Learning*" e da sua forma abreviada "ML", estudou-se a possibilidade de usar termos relacionados com a IA, no entanto, percebeu-se que a direção dos resultados se afastava do foco do trabalho, uma vez que os artigos referenciavam alguns algoritmos (Meta-Heurísticas) como sendo técnicas de IA, e apenas retratavam a aplicação dos mesmos no Escalonamento e não abordavam os algoritmos de ML apresentados na revisão bibliográfica deste documento.
- → Para todas as Palavras-Chave aplica-se o termo *TOPIC*, o que conduz a pesquisa para os campos mais importantes, neste caso título, resumo e palavras-chave.

Tabela 16- Pesquisa por Palavras-Chave

Base de Dados	*Query*	Resultados
Web of Science	*TOPIC ((“Scheduling Problem” OR “Scheduling Problems” OR “Production Scheduling”) AND (“Machine Learning” OR “ML”))*	383

Para analisar e avaliar quantitativamente as publicações utilizaram-se as seguintes dimensões: evolução temporal das publicações e citações, áreas de estudo com mais publicações e coocorrência das palavras-chave utilizadas. O tratamento e mapeamento de dados foi realizado recorrendo ao software *VOSViewer*, nomeadamente para avaliar esta última dimensão [259].

A Figura 39 revela a evolução do interesse no tema desta dissertação entre 1997 a 2023. O artigo que aparece como publicado em janeiro de 2024 já foi aceite e está disponível online desde setembro de 2023. É importante mencionar que os números de publicações permanecem relativamente baixos até ano de 2017, no entanto, a partir de 2018, há um notável aumento, que se intensifica nos últimos três anos, indicando assim uma crescente relevância deste tema. O ano mais produtivo foi 2020 com 75 publicações.

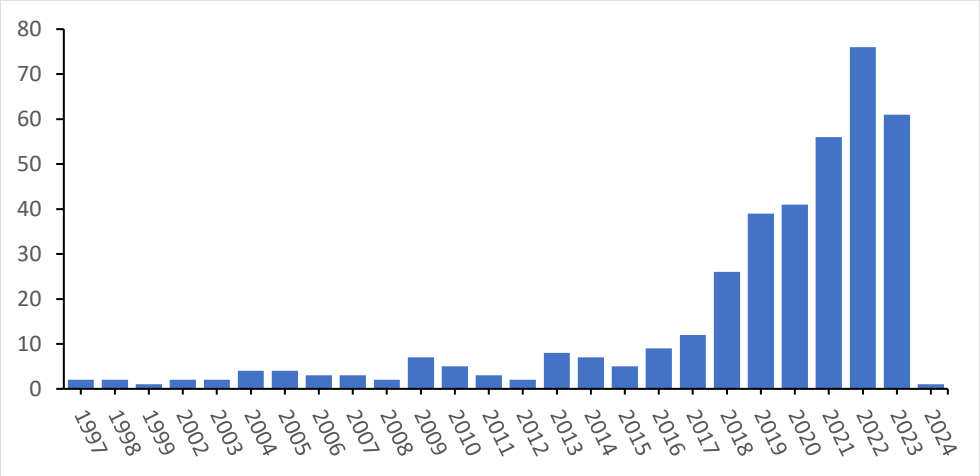


Figura 39- Distribuição temporal das publicações entre 1997 e dezembro de 2024

A análise das áreas de investigação a que as publicações se associam é importante para caracterizar o campo de investigação. Os resultados apresentados na Figura 40 mostram que, das categorias do *Web of Science*, a “*Ciência da Computação, Inteligência Artificial”* é a mais frequente, o que se enquadra no tema pois compreende áreas que fornecem uma base teórica e prática para o desenvolvimento de algoritmos. A *”Engenharia Elétrica e Eletrónica”* e a “*Pesquisa Operacional e Ciência da Gestão”* posicionam-se de seguida, sugerem áreas mais específicas, e, no caso desta última, sublinha a importância da otimização no tema deste trabalho. A diversidade de áreas de investigação evidencia a natureza multidisciplinar do tema.

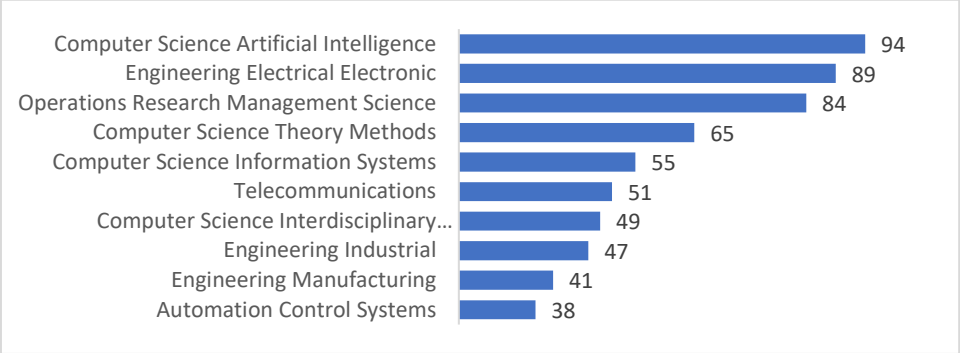


Figura 40- As dez principais áreas de investigação das publicações

A Figura 41 apresenta o mapa representativo da coocorrência e relevância das palavras-chave de cada publicação, que apresenta cinco *clusters*. "*Machine Learning*" é a palavra com maior destaque, seguida de "*optimization*", "*scheduling*" e "*algorithm*", todas praticamente no mesmo nível. Estes são termos com grande relevância pois a aplicação de algoritmos de ML para resolver Problemas de Escalonamento é o cerne do trabalho. A tendência é aplicá-los para melhorar o desempenho das técnicas de otimização. No patamar seguinte surge "*heuristics*", que é crucial no contexto do Escalonamento, porque é comum utilizar estas técnicas para resolver Problemas complexos de Otimização. Por fim, numa perspetiva de análise às tendências e oportunidades futuras, expressões como *"genetic algorithm"*, *"genetic programming"* e *"reinforcement learning"* são relevantes porque são abordagens que se tornaram tendências recentes de investigação por parte da comunidade científica. De maneira geral, o mapa mostra que todos os conceitos estão direta ou indiretamente relacionados.

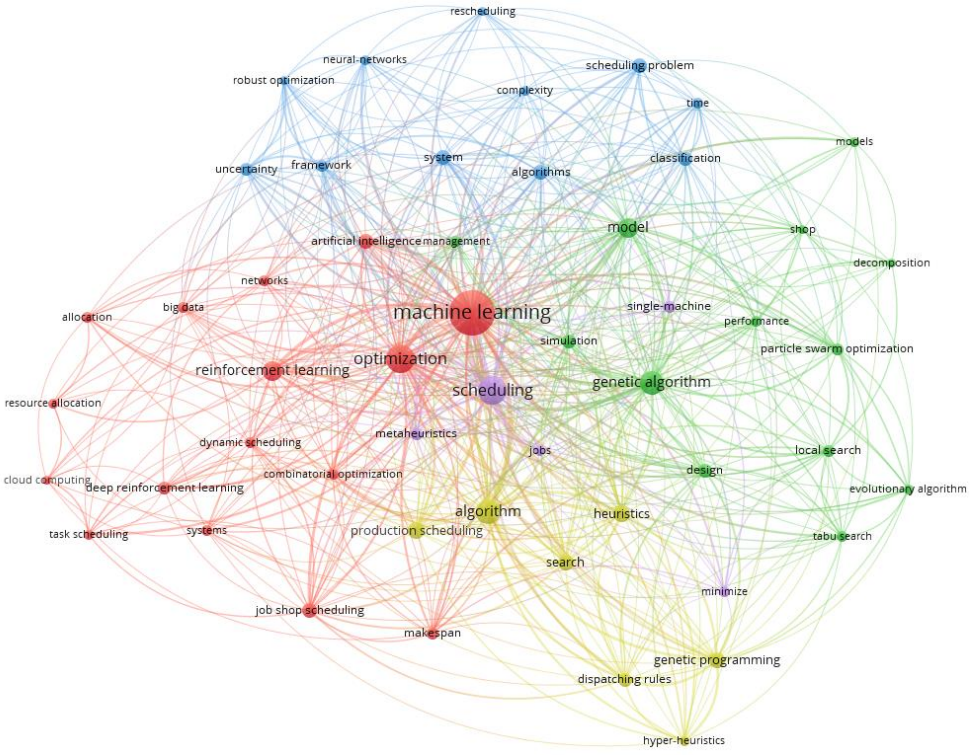


Figura 41- Mapa de Coocorrência de Palavras-chave nos artigos sobre *Machine Learning* e Escalonamento das Operações

Revisão do Estado da Arte - Estudo de Casos

O facto de o ML capacitar sistemas a aprender e melhorar tem transformado a maneira como abordamos e resolvemos problemas complexos. No contexto do Escalonamento, observa-se uma crescente aplicação dos tipos de Aprendizagem para resolver problemas de diferentes ambientes de produção com características dinâmicas e flexíveis, mas também para melhorar o desempenho das técnicas de otimização, como as Meta-Heurísticas.

Reinforcement Learning para resolver Problemas de Escalonamento

O RL tem-se destacado pelo seu extraordinário progresso recente devido à capacidade de os Agentes serem treinados e melhorarem as suas habilidades ao longo do tempo [293]. Neste sentido, diversos autores usam-na para resolver problemas de Escalonamento e perceber se obtêm melhores resultados que com as técnicas atuais, isto é, que os métodos de otimização exatos e aproximados tradicionais.

Cunha et al. em [192] desenvolveram um sistema de Escalonamento completo capaz de fornecer a solução otimizada para o problema de *Job Shop*. O sistema é ativado através de uma instância do problema a ser resolvido. As informações contidas no arquivo são convertidas pelo decodificador em objetos viáveis no sistema, isto é, objetos representativos das máquinas, operações, entre outros. Na fase seguinte, o Agente inteligente, que é composto pelo ambiente de treino e pela técnica de aprendizagem, é responsável por resolver o problema. O ambiente de treino utiliza como base os standards definidos pelo *Gym*, enquanto o algoritmo de aprendizagem implementado foi o PPO, que é responsável por decidir quais operações atribuir e quando. Após todas as tarefas estarem atribuídas e com os tempos definidos, o Agente decide se é melhor repetir o processo de atribuição com base no novo conhecimento adquirido, ou se a política atual tem qualidade suficiente para calcular o plano otimizado final. Após ser treinado resolvendo milhões de vezes diversos problemas, o modelo foi capaz de obter soluções com qualidade bastante superior à de regras de despacho como SPT e LPT. Quando comparado com as Meta-Heurísticas, estas são capazes de apresentar soluções ótimas melhores em alguns minutos, mas o tempo de execução do Agente Inteligente é de poucos segundos, o que representa, de forma incontestável, uma melhoria significativa.

Em [294], é proposta uma arquitetura *Actor-Critic* para, através de DRL, resolver problemas *Job Shop* com acidentes inesperados, como avarias de máquinas. Neste trabalho, existem vários Agentes, cada um associado a uma máquina específica. O modelo compreende duas partes: a rede de atores, que recorre à rede neural convolucional para aprender o comportamento contínuo do Agente que pode obter a máxima recompensa acumulada, e a rede crítica, no qual o Agente avalia o valor da informação recebida e dá uma aproximação que lhe permite saber se o estado que enfrentou é bom ou não. Cada ação são regras de despacho, em que o Agente a seleciona e aloca o trabalho que a satisfaz. O método de treino paralelo proposto combina atualização assíncrona e gradiente de política determinístico profundo. Quando comparado com técnicas de otimização onde o objetivo é minimizar o *makespan*, o método proposto consegue apresentar melhores resultados que as regras de despacho simples, mas está num

nível inferior que as Meta-Heurísticas híbridas. Mesmo assim, destaca-se porque consegue lidar com paragens inesperadas de forma eficiente.

Algoritmos de ML para Melhorar o Desempenho das Técnicas de Otimização Atuais

Considerando a conexão entre algoritmos de RL e Heurísticas e Meta-Heurísticas em Problemas de Escalonamento, é importante mencionar o trabalho desenvolvido em [214]. No algoritmo *Iterated Greedy*, com o objetivo de melhorar a capacidade de exploração do algoritmo, os autores desenvolvem um mecanismo de perturbação que utiliza um algoritmo *Q-Learning* para determinar qual é o operador que deve ser aplicado em cada etapa do processo de pesquisa. Na fase de destruição do algoritmo original, o mecanismo de perturbação consiste na remoção aleatória de uma quantidade de trabalhos da solução atual, sendo o valor desta quantidade conhecido a priori e permanecendo constante durante todo o processo. Na alteração proposta pelos autores, a aplicação de cada operador, isto é, o valor da quantidade de trabalhos a remover, varia em cada iteração. Com base na recompensa imediata do ambiente e no histórico de desempenho dos operadores, o algoritmo atribui um crédito a cada operador. Dependendo do estado atual de procura e do crédito de cada operador, o próximo operador é selecionado para ser aplicado. A aplicação em instância de problemas de *Flow Shop* de Permutação mostraram que o mecanismo de perturbação proposto aumenta consideravelmente a capacidade de exploração do algoritmo. Relativamente à qualidade das soluções, o algoritmo consegue-se destacar por ser capaz de obter melhores soluções que os demais. Por sua vez, nos tempos de CPU, requer mais para conjuntos de instâncias mais pequenas, o que, numa curta escala de tempo, não é tão prejudicial. Para instâncias de maiores dimensões, já é o algoritmo que exige menos tempo de CPU, exceto que o algoritmo onde a quantidade de trabalhos a remover é fixo e igual a 1, mas, em contrapartida, consegue soluções com qualidade superior.

Em [295], os autores desenvolvem o *Memetic Algoritm*, baseado em ML, onde as técnicas introduzidas têm como objetivo melhorar o seu desempenho. Os autores implementaram o método de classificação para, através da análise de dados históricos acumulados durante o processo de pesquisa, selecionar as soluções representativas. A ideia foi dividir o conjunto de soluções não dominadas em vários grupos nos quais a distância entre soluções adjacentes é menor que um determinado limite, o que permite assim que a solução menos aglomerada em cada grupo fosse considerada a solução representativa. Num segundo momento, recorreu-se a um método de aprendizagem estatístico para determinar a quantidade de trabalhos consecutivos a remover da solução atual. Estas alterações conseguiram melhorar o desempenho do algoritmo porque se evitou a duplicação da pesquisa em soluções semelhantes e se restringiu a pesquisa local à procura exclusiva a regiões promissoras.

Outros Casos de Estudo Relevantes

Para além dos artigos apresentados anteriormente, existem outros que desempenham um papel de extrema importância no crescimento da investigação da aplicação de algoritmos de ML para resolver problemas de Escalonamento. A Tabela 17 apresenta publicações que se destacam pela aplicação dos algoritmos em si, mas também revisões da literatura que permitem perceber o estado atual da temática.

Tabela 17- Outros artigos relevantes sobre as duas áreas importantes em análise

Autores	Descrição do Trabalho
[296]	Este estudo apresenta uma abordagem para melhorar a eficiência da Heurística ATCS-BATCS (β) através da aplicação de técnicas de ML. Esta Heurística, que é comum para resolver Problemas de Escalonamento, possui três parâmetros de controlo que influenciam a qualidade da solução obtida e que podem ser determinados a partir de uma abordagem *multi-start*. Dada uma instância do problema e parâmetros heurísticos específicos, o algoritmo *Artificial Neural Newtworks* foi usado para prever o desempenho da Heurística. Esta previsão permite realizar uma pesquisa em grade, que é característico da abordagem mencionada, num conjunto menor e promissor de parâmetros. Para lidar com a incerteza inerente às previsões dos parâmetros, foram aplicadas múltiplas configurações utilizando classificação baseada em previsão. Esta abordagem demonstrou ser eficiente, nomeadamente por apenas necessitar de uma fração de tempo de computação para garantir a pesquisa em grade completa.
[297]	Neste artigo, utiliza-se os algoritmos *Neural Networks* e *Inductive Decision Trees* para estimar parâmetros em Heurísticas num Problema de Escalonamento. A regra de despacho do Custo Aparente de Atraso tem em consideração as regras de menor tempo de processamento ponderado e de menor folga. Esta última é dimensionada pela constante *k*, que é sensível ao valor que lhe é atribuído. Na estimativa antecipada deste parâmetro, a fase de treino do algoritmo *Inductive Decision Tree* demorou apenas alguns segundos, enquanto a da abordagem *Neural Network* levou vários minutos, o que evidencia a capacidade da primeira técnica superar ligeiramente a segunda.
[298]	Numa vertente teórica e de revisão, este estudo aborda as técnicas utilizadas para os aprimorar os Algoritmos Evolutivos, onde se incluem as técnicas de ML. Algumas das técnicas de ML detalhadas são a *Regression, Clustering Analysis, Principle Component Analysis, Support Vector Machine, Artificial Neural Networks* e *Opposition-Based Learning*. Estas técnicas são utilizadas para obter melhores soluções iniciais, reduzir o processo de avaliação de funções de desempenho para obter soluções de qualidade, aprimorar operadores de mutação, recombinação e seleção, reutilizar e incorporar, no processo de pesquisa o conhecimento adquirido em experiências anteriores, e ajustar parâmetros característicos.
[299]	Neste estudo, um algoritmo de Aprendizagem Supervisionada, o *Random Search*, foi utilizado para estimar fatores de ponderação na função objetivo para Problemas de Otimização Multiobjetivo. Numa fase inicial, a partir de informações como tempo de entrega, tempo de processamento e custo de *setup*, recorre-se a um algoritmo exato (problemas de pequena escala) ou ao SA (problemas de grande escala) para resolver problemas de máquinas paralelas e obter os dados de saída. Posteriormente, estas informações, juntamente com

Autores	Descrição do Trabalho
	os respetivos fatores de ponderação, foram inseridas para treinar e construir o modelo de aprendizagem. Concluído o treino, novos dados de teste foram utlizados para o modelo estimar o fator de ponderação. Os resultados obtidos através desta estrutura demonstraram que a mesma é significativamente eficaz.
[300]	Numa forma de revisão da literatura, este trabalho aborda as várias categorias de Escalonamento Inteligente, onde se inclui o ML. *Case-Based Reasoning, Inductive Learning* e *Neural Networks* os principais tipos de algoritmos detalhados. Para cada um elabora-se uma pequena descrição, apresentam-se exemplos de aplicações e enumeram-se as principais forças e fraquezas. Devido à capacidade destas técnicas, os autores terminam realçando a importância do ML e o seu potencial crescimento no contexto do Escalonamento Inteligente.

Os casos de estudo apresentados na Tabela 17 evidenciam assim a aplicação de vários tipos de aprendizagem de ML para resolver Problemas de Escalonamento, o que é representativo da capacidade que têm de ajudar a superar as limitações dos métodos tradicionais de otimização.

Caminhos Futuros

O uso de algoritmos de ML tem-se mostrado promissor na resolução de Problemas de Escalonamento das Operações. A revisão da literatura revelou avanços significativos, incluindo o uso de RL para os resolver de forma autónoma, e a sinergia entre diferentes tipos de aprendizagem e as técnicas de otimização tradicionais, como Heurísticas e Meta-Heurísticas, onde o principal objetivo é melhorar os métodos atuais para alcançar resultados superiores.

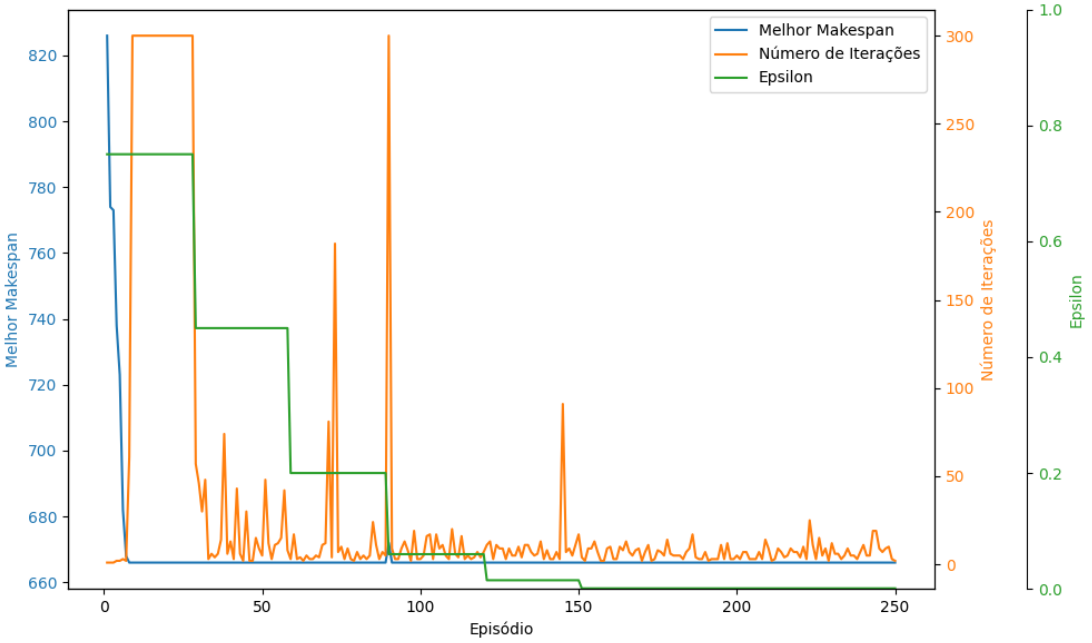
A crescente pesquisa por interações entre RL e Meta-Heurísticas, seja através de Agentes de política completamente autónomos, ou, inclusive, na exploração mais eficiente do espaço de soluções, comprova o destaque dado ao RL pelo seu extraordinário progresso recente, o que o coloca como uma das áreas de investigação mais ativas e promissoras. Tendo consideração o sucesso dos mecanismos baseados em algoritmos de RL, como o *Q-Learning*, uma direção de pesquisa pode ser adaptá-los ou criar outros, baseados noutros algoritmos, para outras Meta-Heurísticas que não tem sido um foco de estudo neste tema, como o SA, o TS, os Algoritmos Genéticos, entre outros. Um primeiro passo pode ser, por exemplo, utilizá-los para ajustar dinamicamente a temperatura do SA como base no comportamento passado e nas características do problema, ou ajustar a lista Tabu dinamicamente, evitando ótimos locais.

Relativamente a outros tipos de aprendizagem, a sua incorporação em abordagens capazes de resolver Problemas de Escalonamento não tem tido tanta aplicação, no entanto, quando se abordam ambientes reais, caracterizados por constantes mudanças, estes podem ser explorados de forma eficiente. Pode ser interessante recorrer à Aprendizagem Supervisionada para melhorar a fase inicial de treino dos Agentes, fornecendo noções sobre ações mais promissoras para evitar que explorem ações ineficientes. Noutra vertente, e em função da técnica de classificação baseada em previsão de desempenho aplicada em exemplos

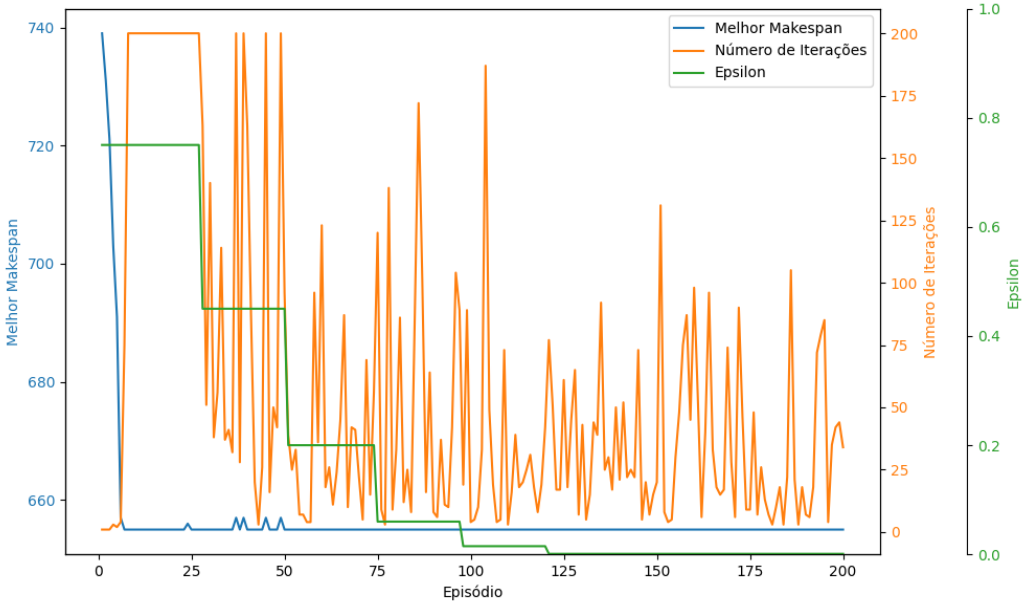
apresentados neste estudo, estender a sua aplicação para prever o desempenho de múltiplos algoritmos podia permitir a seleção e posterior execução apenas dos melhores.

Apêndice C

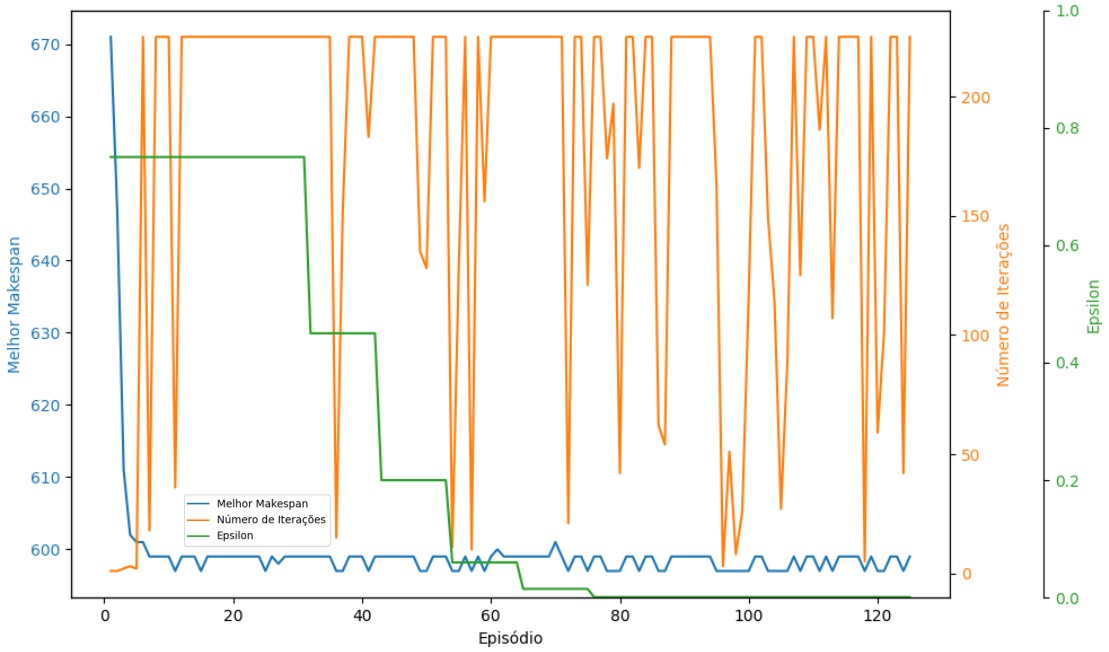
Instância la01



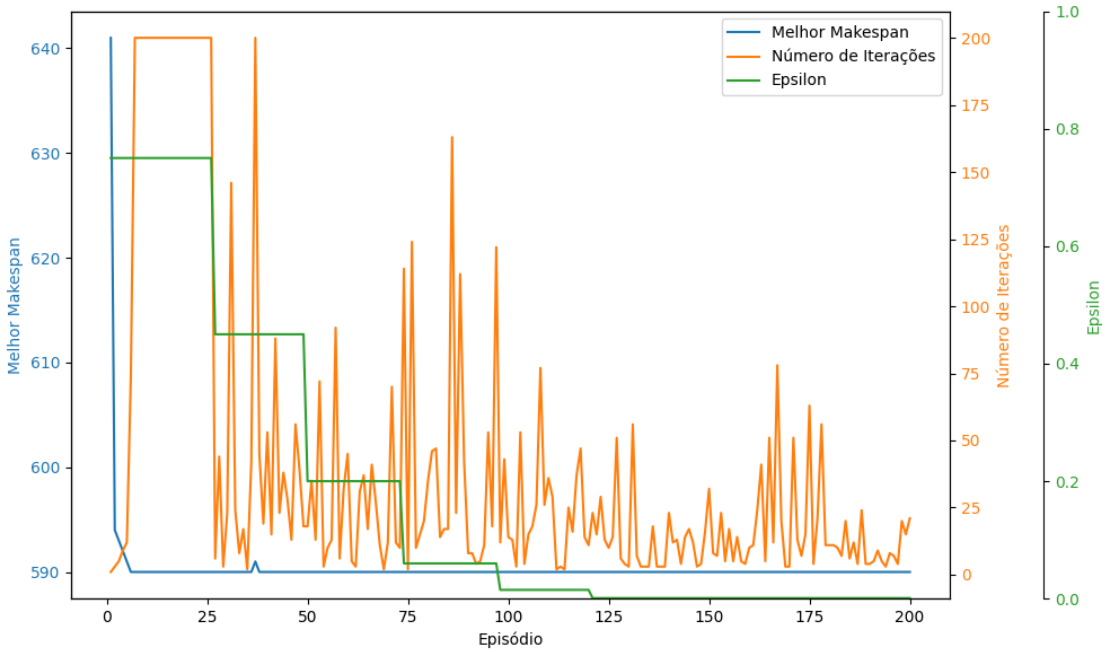
Instância la02



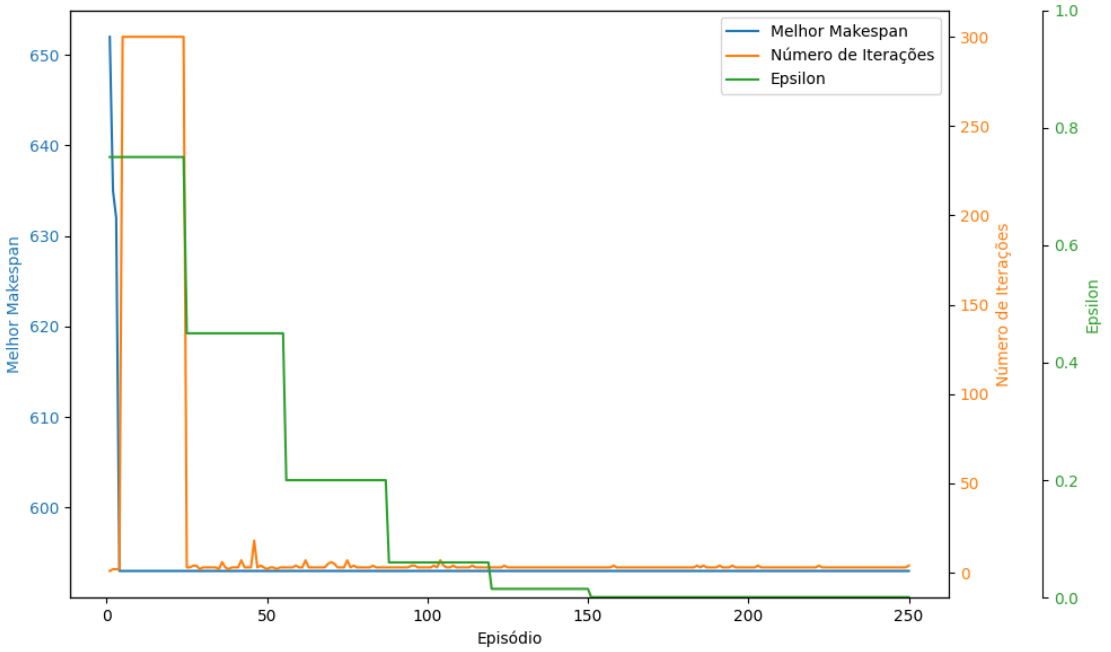
Instância la03



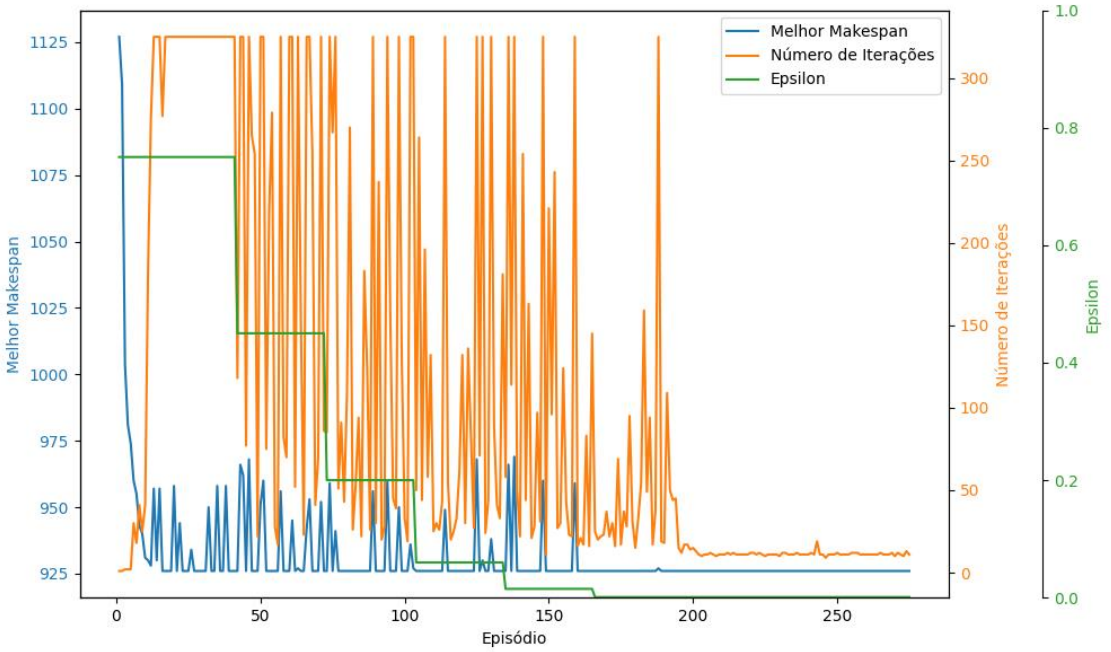
Instância la04



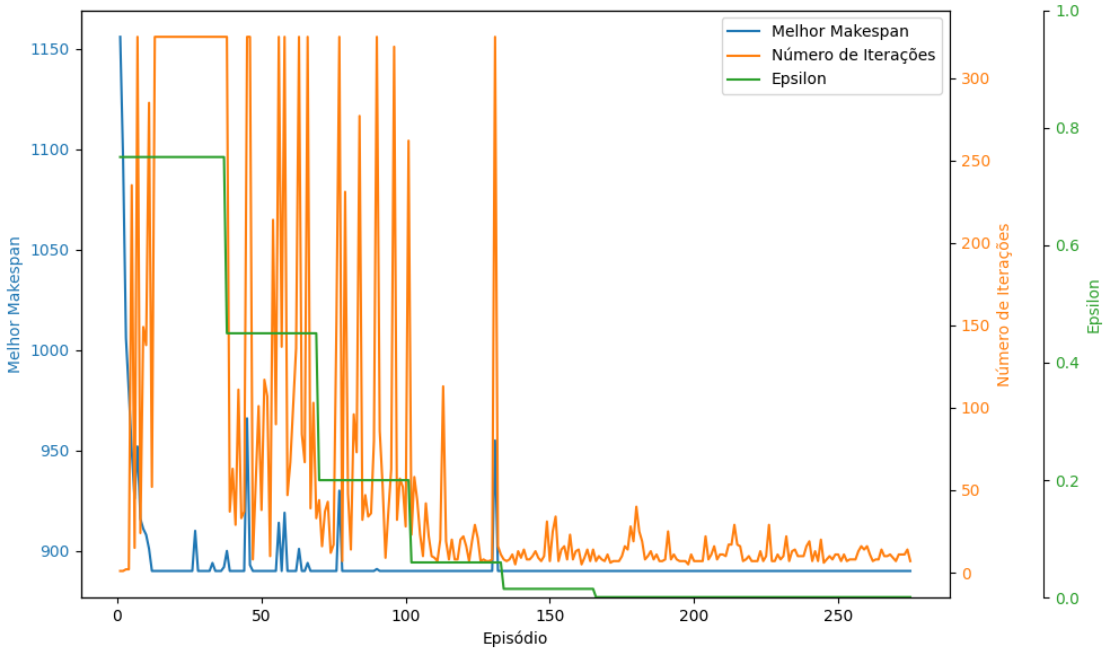
Instância la05



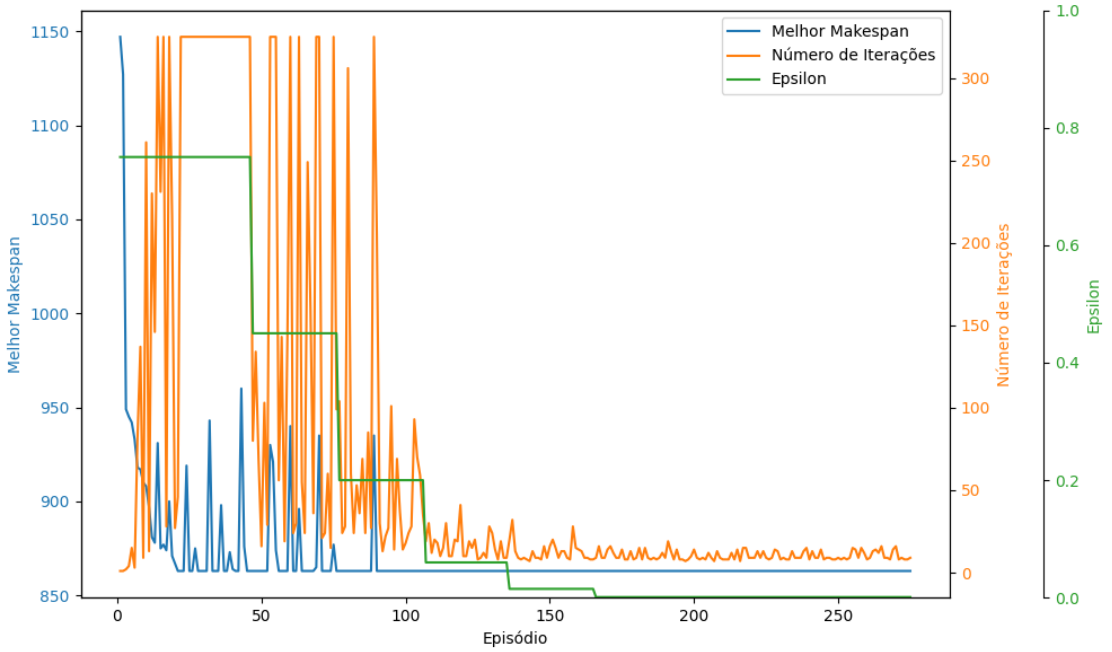
Instância la06



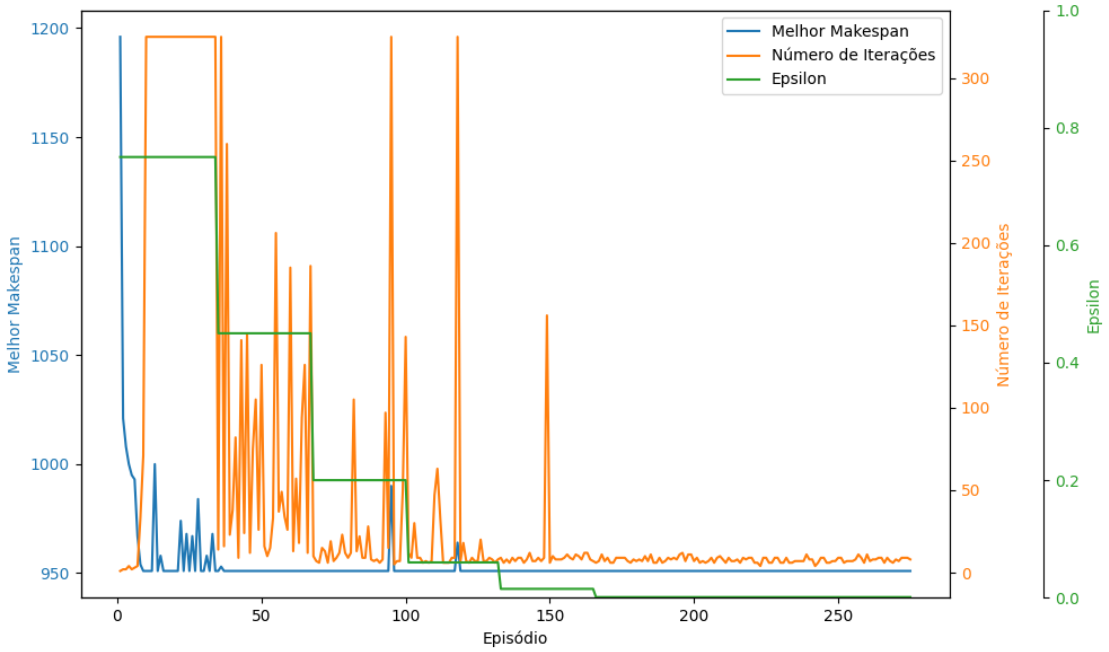
Instância la07



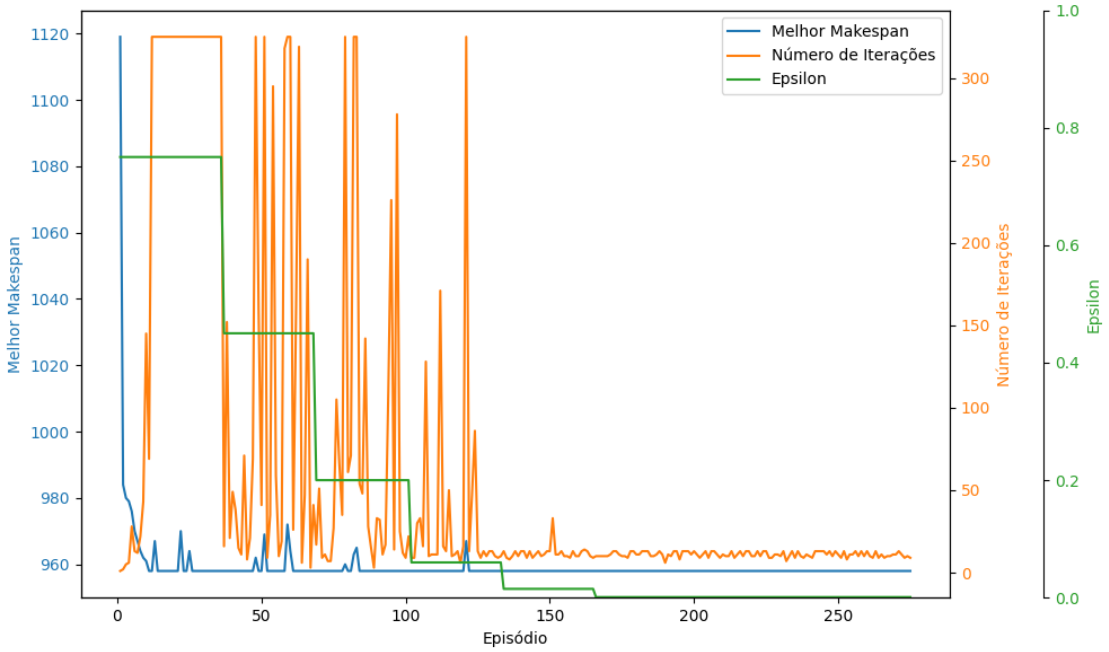
Instância la08



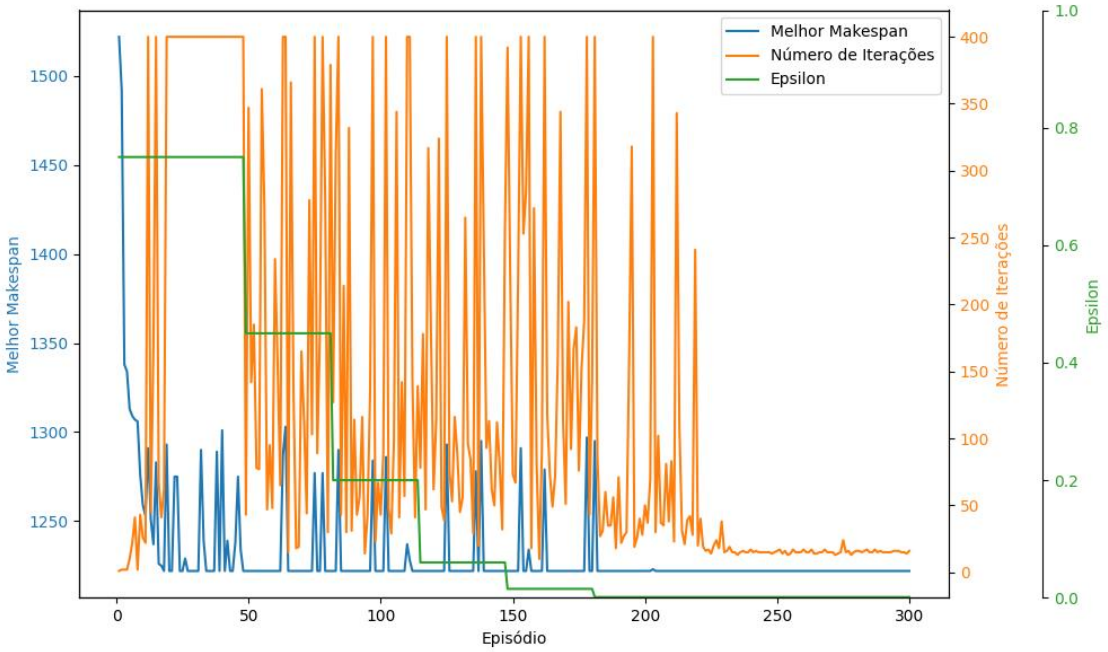
Instância la09



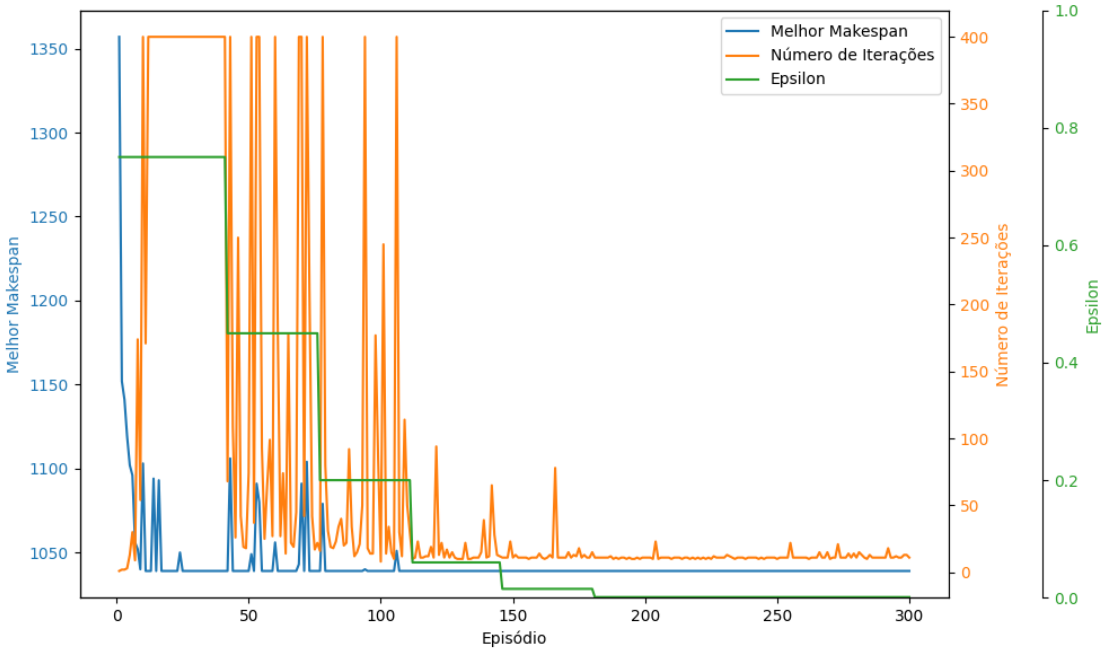
Instância la10



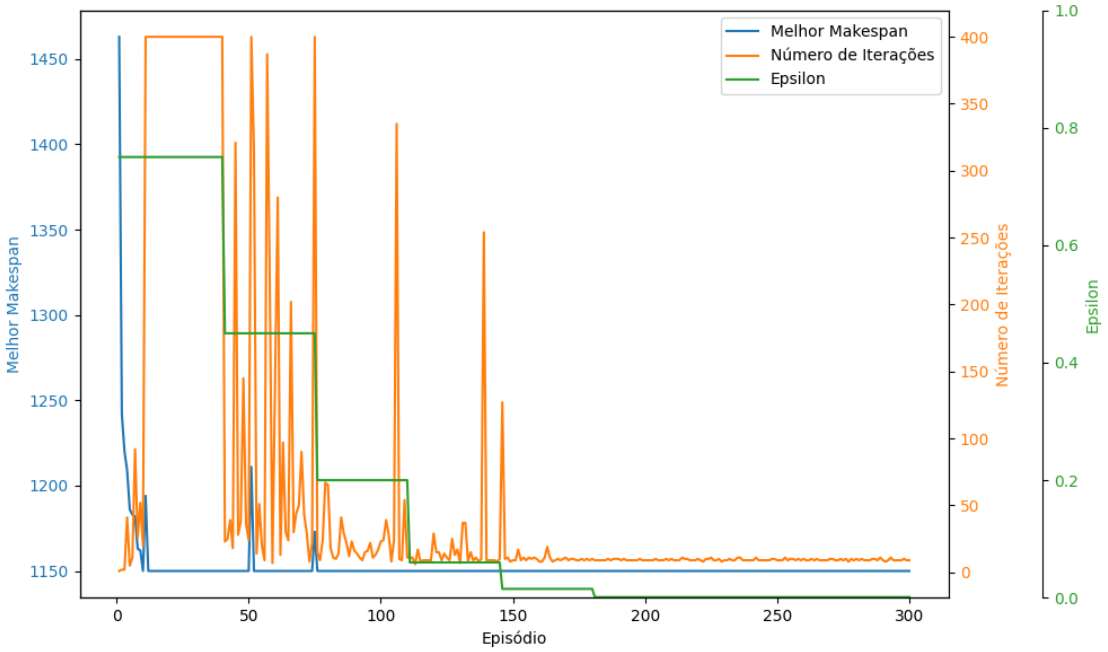
Instância la11



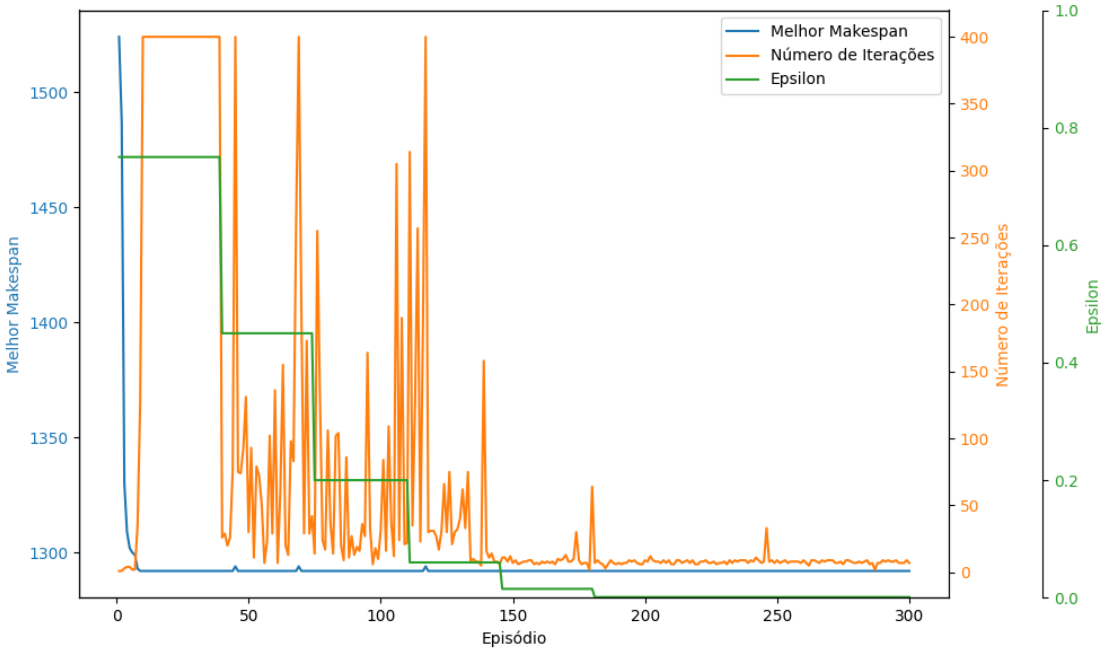
Instância la12



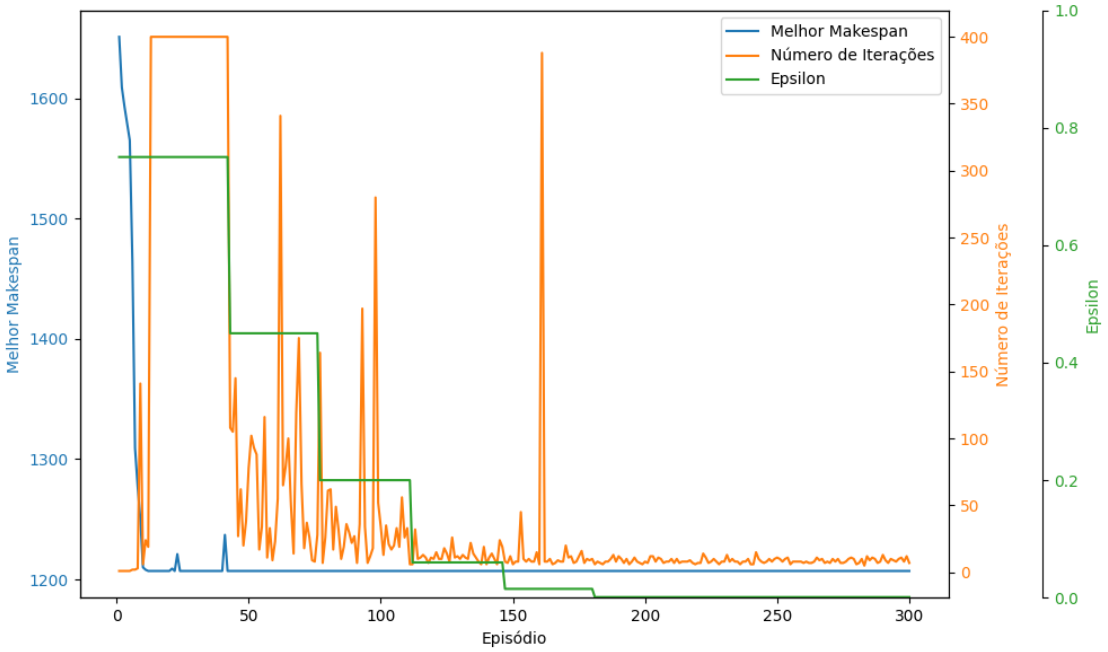
Instância la13



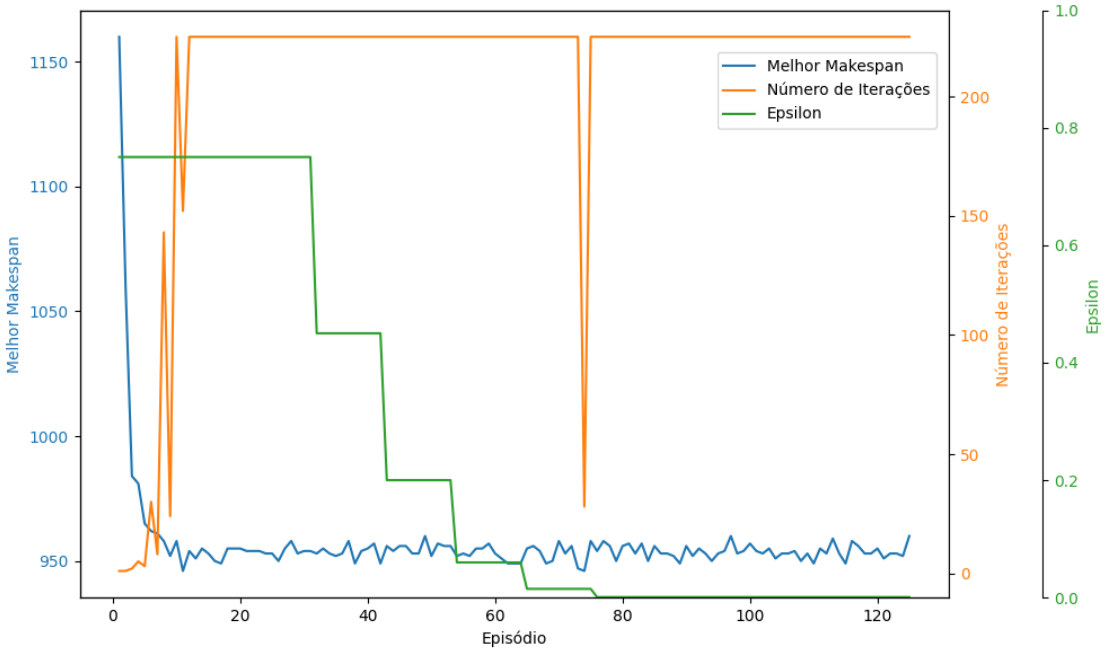
Instância la14



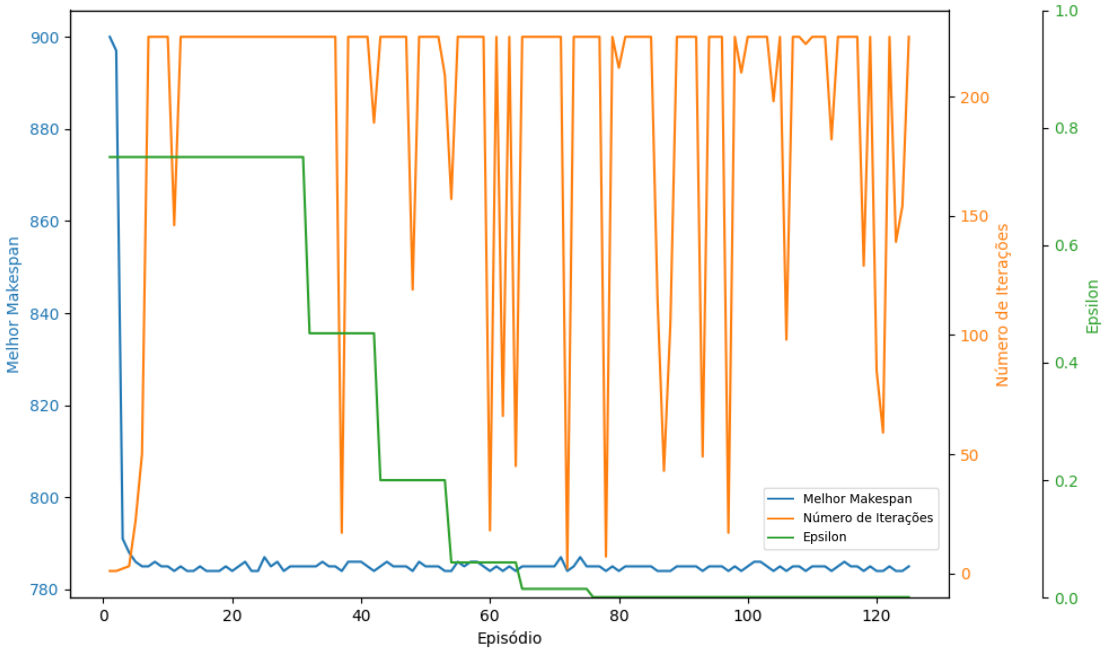
Instância la15



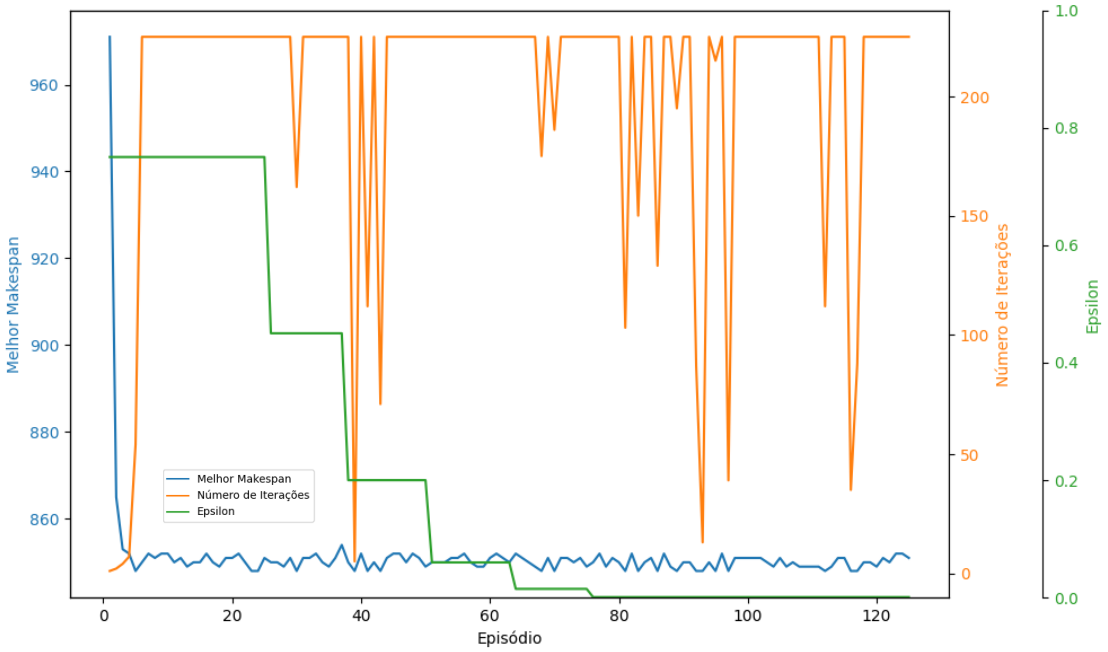
Instância la16



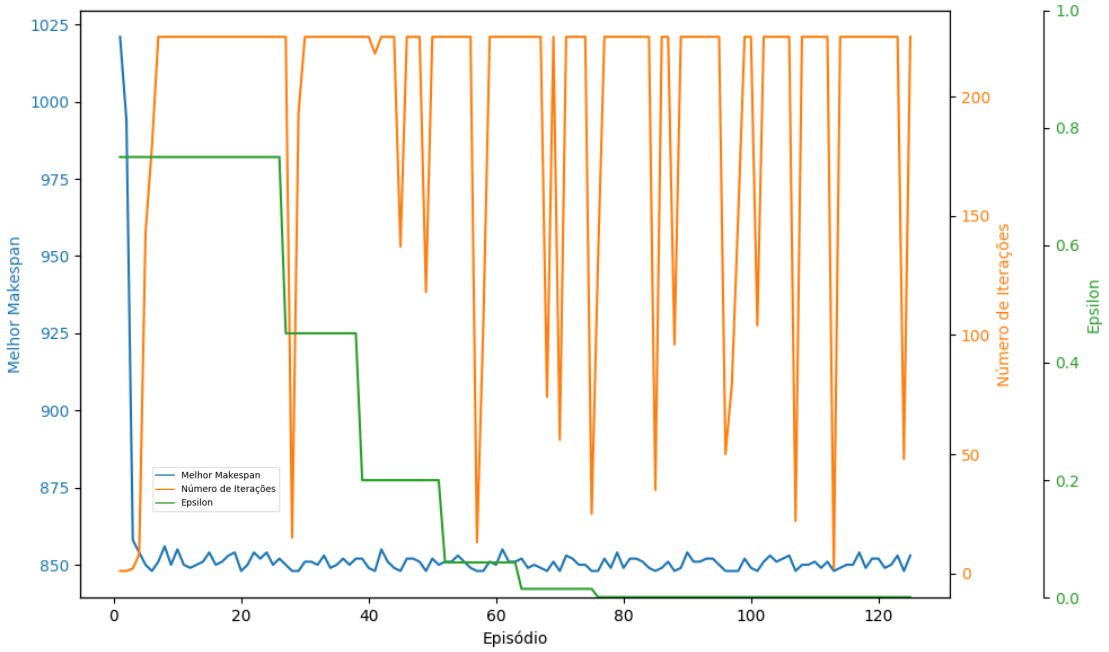
Instância la17



Instância la18



Instância la19



Instância la20