HÍPER-HEURÍSTICAS COM APRENDIZAGEM

Diamantino Fernando Madureira Falcão

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Área de Especialização em
Tecnologias do Conhecimento e Decisão

Orientador: Doutora Ana Maria Dias Madureira Pereira

Co-orientador: Doutor Ivo André Soares Pereira

Júri:

Presidente:

Doutor João Paulo Jorge Pereira, Professor Adjunto, Instituto Superior de Engenharia do Porto, Departamento de Engenharia Informática

Vogais:

Doutor Eduardo José Solteiro Pires, Professor Auxiliar, Universidade de Trás-os-Montes e Alto Douro, Escola de Ciências e Tecnologia, Departamento de Engenharias

Doutora Ana Maria Dias Madureira Pereira, Professora Coordenadora, Instituto Superior de Engenharia do Porto, Departamento de Engenharia Informática

Doutor Ivo André Soares Pereira, Instituto Superior de Engenharia do Porto, Departamento de Engenharia Informática

Porto, Outubro 2014

«À minha família e amigos»

Resumo

A otimização nos sistemas de suporte à decisão atuais assume um carácter fortemente interdisciplinar relacionando-se com a necessidade de integração de diferentes técnicas e paradigmas na resolução de problemas reais complexos, sendo que a computação de soluções ótimas em muitos destes problemas é intratável. Os métodos de pesquisa heurística são conhecidos por permitir obter bons resultados num intervalo temporal aceitável. Muitas vezes, necessitam que a parametrização seja ajustada de forma a permitir obter bons resultados.

Neste sentido, as estratégias de aprendizagem podem incrementar o desempenho de um sistema, dotando-o com a capacidade de aprendizagem, por exemplo, qual a técnica de otimização mais adequada para a resolução de uma classe particular de problemas, ou qual a parametrização mais adequada de um dado algoritmo num determinado cenário.

Alguns dos métodos de otimização mais usados para a resolução de problemas do mundo real resultaram da adaptação de ideias de várias áreas de investigação, principalmente com inspiração na natureza - Meta-heurísticas. O processo de seleção de uma Meta-heurística para a resolução de um dado problema é em si um problema de otimização.

As Híper-heurísticas surgem neste contexto como metodologias eficientes para selecionar ou gerar heurísticas (ou Meta-heurísticas) na resolução de problemas de otimização *NP-difícil*.

Nesta dissertação pretende-se dar uma contribuição para o problema de seleção de Meta-heurísticas e respetiva parametrização. Neste sentido é descrita a especificação de uma Híper-heurística para a seleção de técnicas baseadas na natureza, na resolução do problema de escalonamento de tarefas em sistemas de fabrico, com base em experiência anterior. O módulo de Híper-heurística desenvolvido utiliza um algoritmo de aprendizagem por reforço (*Q-Learning*), que permite dotar o sistema da capacidade de seleção automática da Meta-heurística a usar no processo de otimização, assim como a respetiva parametrização.

Finalmente, procede-se à realização de testes computacionais para avaliar a influência da Híper-Heurística no desempenho do sistema de escalonamento AutoDynAgents. Como conclusão genérica, é possível afirmar que, dos resultados obtidos é possível concluir existir vantagem significativa no desempenho do sistema quando introduzida a Híper-heurística baseada em *Q-Learning*.

Palavras-chave: Híper-heurística, Aprendizagem Automática, Q-Learning, Otimização, Escalonamento de tarefas, Meta-heurísticas, Sistemas Multiagente.

Abstract

Optimization in current decision support systems has a highly interdisciplinary nature related with the need to integrate different techniques and paradigms in solving real-world complex problems. Computing optimal solutions in many of these problems are unmanageable. Heuristic search methods are known to obtain good results in an acceptable time interval. Often, parameters need to be adjusted to allow good results.

In this sense, learning strategies can enhance the performance of a system, providing it with the ability to learn, for instance, the most suitable optimization technique for solving a particular class of problems, or the most suitable parameterization of a given algorithm on a given scenario.

Some of the most used optimization methods for solving real world problems resulted from the adaptation of ideas from several areas of research, especially the ones inspired by nature - Metaheuristics. The process of selecting a Metaheuristics for solving a given problem is by itself an optimization problem.

The Hyper-heuristics arise in this context as efficient methodologies for selecting or generating heuristics (or Metaheuristics) to solve *NP-hard* optimization problems.

This thesis aims to make a contribution to the problem of selection of Metaheuristics and respective parameterization. In this sense, the specification of a Hyper-heuristic is describes for the selection of techniques based in nature, in solving the problem of scheduling in manufacturing systems, based on previous experience. The developed Hyper-heuristic module uses a reinforcement learning algorithm (Q-Learning), which enables the system with the ability to autonomously select the Metaheuristics to use in optimization process as well as the respective parameters.

Finally, a computational study was carried out to evaluate the influence of the Hyper-heuristics on the performance of the AutoDynAgents system. As a general conclusion, we can say from the results obtained that there is significant advantage in using the system with the *Q-Learning* based Hyper-heuristic.

Keywords: Hyper-heuristics, Machine Learning, Q-Learning, Optimization, Scheduling, Metaheuristics, Multi-Agent Systems.

Agradecimentos

Gostaria de agradecer a todas as pessoas que de alguma forma contribuíram para a realização deste trabalho, em especial:

À minha orientadora, Doutora Ana Maria Dias Madureira Pereira, por toda a ajuda prestada, pela disponibilidade que teve sempre e sobretudo pelo enorme trabalho de revisão desta dissertação.

Ao meu co-orientador, Mestre Ivo André Soares Pereira, por toda a ajuda prestada, pelo vasto material bibliográfico disponibilizado e revisão desta dissertação.

Ao Grupo de Investigação em Engenharia do Conhecimento e Apoio à Decisão – GECAD do Instituto Superior de Engenharia do Porto, por todos os meios disponibilizados para a realização deste trabalho.

Aos meus colegas de curso, por todo o apoio, sugestões, críticas, companheirismo e amizade que demonstraram ao longo dos últimos dois anos.

Ao corpo Docente e não Docente do Instituto Superior de Engenharia do Porto.

À minha família, pela dedicação, determinação, força e coragem que sempre me transmitiram, não só durante a realização deste trabalho, mas em todas as etapas da minha vida, nomeadamente no meu percurso académico.

A todos os meus amigos, que nunca se esqueceram de mim, apesar de muitas vezes ter negado a minha presença em eventos importantes.

A todos, o meu muito obrigado!

Índice

1 Introdução 1

1.1 Objetivos e Principais Contribuições 2

1.2 Estrutura do Documento 3

2 O Problema de Escalonamento 5

2.1 Conceitos e Definições 5

2.2 Classificação do Problema de Escalonamento 6

2.3 Abordagens de Resolução 7

2.3.1 Regras de Prioridade 7

2.3.2 Pesquisa Local 8

2.3.3 Meta-Heurísticas 8

2.3.4 Híper-heurísticas 9

2.3.5 Outras Técnicas 9

2.4 Sumário do Capítulo 10

3 Aprendizagem Automática 11

3.1 Perspetiva Histórica 11

3.2 Tipos de Aprendizagem Automática 13

3.2.1 Aprendizagem Supervisionada 14

3.2.2 Aprendizagem não Supervisionada 14

3.2.3 Aprendizagem por Reforço 15

3.3 Técnicas de Aprendizagem Automática 15

3.3.1 Árvores de Decisão (*Decision Trees*) 16

3.3.2 Aprendizagem Baseada em Instâncias (*Instance Based Learning*) 16

3.3.3 Aprendizagem de Regras (*Rules Learning*) 17

3.3.4 Redes Neuronais Artificiais (*Artificial Neural Networks*) 18

3.3.5 Redes Bayesianas (*Bayesian Networks*) 19

3.3.1 Support Vector Machines 20

3.3.2 Agrupamento de dados (*Clustering*) 21

3.3.3 Q-Learning 22

3.4 Avaliação das Técnicas de Aprendizagem Automática 23

3.4 Sumário do Capítulo 24

4 Meta-Heurísticas e Híper-Heurísticas 25

4.1 Perspetiva Histórica 27
4.2 Meta-Heurísticas 28
4.2.1 Algoritmos Genéticos 29
4.2.2 Colónia Artificial de Abelhas 30
4.2.3 Otimização por Colónia de Formigas 32
4.2.4 Particle Swarm Optimization 33
4.2.5 Pesquisa Tabu 35
4.2.6 Simulated Annealing 36
4.3 Metodologia de Seleção de Heurísticas 37
4.3.1 Abordagens para Seleção de Heurísticas 38
4.4 Áreas de Aplicação 40
4.4.1 Abordagens Offline 41
4.4.2 Abordagens Online 41
4.5 Sumário do Capítulo 42
5 Sistema AutoDynAgents e o Módulo da Híper-Heurística 43
5.1 Sistemas Multiagente 43
5.2 Sistema AutoDynAgents 45
5.2.1 Estrutura do Sistema AutoDynAgents com a Híper-heurística 47
5.2.1.1 Interface Gráfica 48
5.2.1.2 Módulo da Híper-heurística 50
5.2.1.3 Módulo de Escalonamento 50
5.3 Módulo da Híper-Heurística 51
5.3.1 Base de Dados 52
5.3.2 Algoritmo Q-Learning 56
5.3.2.1 Critério de Avaliação do Estado 57
5.3.2.2 Exploração e Aproveitamento 57
5.3.2.3 Função de Recompensa 58
5.4 Exemplo Ilustrativo 58
5.5 Sumário do Capítulo 63
6 Estudo Computacional 65
6.1 Inicialização da Base de Dados 66
6.1.1 Resultados Inseridos na Base de Dados 66
6.1.2 Parametrização Inicial 70

6.2 Resultados Computacionais 74
6.2.1 Resultados Obtidos com a Híper-heurística 74
6.2.2 Comparação com Resultados Prévios 77
6.2.3 Comparação com a Técnica de Aprendizagem Raciocínio Baseado em Casos 79
6.3 Sumário do Capítulo 82
7 Conclusões 83
7.1 Resumo 83
7.2 Contribuições 84
7.3 Limitações e Trabalho Futuro 85
7.4 Considerações finais 86
Bibliografia 87

Lista de Figuras

Figura 1 – Representação de uma Árvore de Decisão 16
Figura 2 – Representação de uma Rede Neuronal 18
Figura 3 – Estrutura de uma Rede Bayesiana 20
Figura 4 – Híper-plano de separação ótimo 21
Figura 5 – Classificação das Híper-heurísticas 26
Figura 6 – Ciclo de uma geração de um Algoritmo Genético 30
Figura 7 – Cenário Multiagente 44
Figura 8 – Arquitetura dos agentes do sistema AutoDynAgents 46
Figura 9 – Estrutura do sistema AutoDynAgents com a Híper-heurística 47
Figura 10 – Gráfico comparativo entre o plano previsto e o gerado, e gráfico comparativo dos pesos das tarefas 48
Figura 11 – Gráfico de Gantt do escalonamento 49
Figura 12 – Relatório de escalonamento 49
Figura 13 – Resultado da execução da Meta-heurística 49
Figura 14 – Base de dados para armazenar os resultados obtidos 53
Figura 15 – Comparação do quociente dos valores médios da execução do sistema com a Híper-heurística 76
Figura 16 - Comparação do quociente dos valores médios da execução do sistema, entre os resultados prévios e os resultados obtidos pela Híper-heurística 78
Figura 17 - Comparação do quociente dos valores médios da execução do sistema, entre os resultados obtidos pelo Raciocínio Baseado em Casos e os resultados obtidos pela Híper-heurística 81

Lista de Tabelas

Tabela 1 – Algoritmo Genético 29
Tabela 2 – Algoritmo Colónia Artificial de Abelhas 31
Tabela 3 – Algoritmo Otimização por Colónia de Formigas 32
Tabela 4 – Algoritmo *Particle Swarm Optimization* 34
Tabela 5 – Algoritmo Pesquisa Tabu 35
Tabela 6 - Algoritmo *Simulated Annealing* 37
Tabela 7 – Métodos de seleção e estratégias de aceitação de Heurísticas 38
Tabela 8 – Método de Escalonamento baseado em Meta-heurísticas 51
Tabela 9 – Campos da tabela MetaHeuristicResult 54
Tabela 10 – Campos da tabela ABC 54
Tabela 11 – Campos da tabela ACO 55
Tabela 12 – Campos da tabela GA 55
Tabela 13 – Campos da tabela PSO 55
Tabela 14 – Campos da tabela SA 56
Tabela 15 – Campos da tabela TS 56
Tabela 16 – Estrutura funcional do algoritmo Q-Learning 56
Tabela 17 – Exemplo de um novo problema 58
Tabela 18 – Exemplo de um conjunto de registos 59
Tabela 19 – Exemplo de parametrização ABC 59
Tabela 20 – Exemplo de parametrização SA 59
Tabela 21 – Exemplo de parametrização TS 60
Tabela 22 – Comando SELECT 60
Tabela 23 – Resultados produzidos pelo sistema após duas iterações 60
Tabela 24 – Matriz de recompensa 61
Tabela 25 – Matriz de valores Q 61
Tabela 26 – Resultado gerado após a aplicação da Meta-heurística 61
Tabela 27 – Matriz de recompensa 62
Tabela 28 – Matriz de valores Q 62
Tabela 29 – Resultado inserido na base de dados 62
Tabela 30 – Parametrização final do SA 62
Tabela 31 – Notação das tabelas de resultados computacionais 66
Tabela 32 – Tempos de conclusão e execução das Meta-heurísticas 67
Tabela 33 – Tempos de conclusão e execução para a Meta-heurística ABC 69
Tabela 34 – Parametrizações para instâncias de 10 tarefas 71
Tabela 35 – Parametrizações para instâncias de 20 e 15 tarefas 72
Tabela 36 – Parametrizações para instâncias de 30 e 50 tarefas 73
Tabela 37 – Parametrizações para a Meta-heurística ABC 74
Tabela 38 – Resultados computacionais obtidos para um critério de paragem de 10, 20 e 30 75
Tabela 39 – Resultado do teste *t* de Student para amostras emparelhadas: 10 Corridas *vs.* 20 Corridas, 10 Corridas *vs.* 30 Corridas, 20 Corridas *vs.* 30 Corridas 76

Tabela 40 – Resultados obtidos sem mecanismo de aprendizagem..... 77
Tabela 41 – Resultado do teste *t* de Student para amostras emparelhadas: Híper-heurística *vs.* Resultados Prévios 79
Tabela 42 - Resultados obtidos pelo módulo Raciocínio Baseado em Casos..... 80
Tabela 43 - Resultado do teste t de Student para amostras emparelhadas: Híper-heurística *vs.* Raciocínio Baseado em Casos 81

Capítulo 1

Introdução

Diariamente, pessoas e empresas são desafiadas com problemas de difícil resolução, tais como encontrar o melhor percurso entre locais diferentes, encontrar o melhor horário possível, ou efetuar gestão de recursos. Para muitos destes problemas de Otimização Combinatória, o número de soluções cresce exponencialmente com a dimensão do problema, e encontrar a melhor solução/soluções torna-se difícil. De facto, os problemas de Otimização Combinatória surgem da necessidade de se selecionar, de um conjunto de dados discreto e finito, o melhor subconjunto que satisfaz determinados critérios de natureza económica-operacional (Pereira, 2009, 2014).

Os problemas de Escalonamento são classificados como problemas de Otimização Combinatória sujeitos a restrições, com uma natureza dinâmica e de resolução muito complexa, sendo classificados como *NP-difíceis*, cujos elementos básicos são as máquinas e as tarefas. O Escalonamento visa a afetação no tempo de tarefas a determinadas máquinas, sujeitas a algumas restrições, como, por exemplo, nenhuma máquina poder processar mais do que uma tarefa em simultâneo (Madureira, 2003).

Neste contexto, os diversos métodos heurísticos apresentam-se como ferramentas adequadas para resolver os problemas de Otimização Combinatória, onde se inclui o problema de Escalonamento. De facto, existem muitos métodos de pesquisa baseado em heurísticas. Alguns dos métodos mais usados resultam da adaptação de ideias de várias áreas, principalmente com base na natureza, e designam-se genericamente por Meta-heurísticas. A escolha de uma Meta-heurística para a resolução de um dado problema pode ser vista como um verdadeiro problema de otimização.

A otimização nas aplicações atuais assume um carácter fortemente interdisciplinar relacionando-se com a necessidade de integração de diferentes técnicas e paradigmas na resolução de problemas reais complexos, sendo que a computação de soluções ótimas em muitos destes problemas é intratável. Neste contexto, as Híper-heurísticas apresentam-se como metodologias que permitem selecionar ou gerar heurísticas (ou Meta-heurísticas) para resolver problemas de otimização particularmente difíceis.

A Aprendizagem Automática surge como um complemento para as Híper-heurísticas, oferecendo uma variedade de técnicas enquadradas na Aprendizagem Supervisionada, Aprendizagem não Supervisionada e na Aprendizagem por Reforço, de modo a ser possível introduzir comportamentos inteligentes na escolha e seleção de técnicas de otimização inspiradas na natureza, com o objetivo de resolver problemas de otimização.

Esta dissertação de mestrado foi realizada no âmbito do Projeto de I&D, aprovado pela Fundação para a Ciência e a Tecnologia, AutoDynAgents – *Autonomic Agents with Self-Managing Capabilities for Dynamic Scheduling Support in a Cooperative Manufacturing System* (POCTI/EMEGIN/66848/2006). O AutoDynAgents consiste num Sistema Multiagente para a resolução autónoma, distribuída e cooperativa de problemas de escalonamento sujeitos a perturbações. Este sistema incorpora conceitos da Computação Autónoma e utiliza Meta-heurísticas para a determinação de planos de escalonamento quase-ótimos. Uma vez que o ambiente de atuação do AutoDynAgents é complexo, dinâmico e imprevisível, as questões de aprendizagem tornaram-se imprescindíveis.

Neste contexto, pretende-se com esta dissertação a realização de um estudo de métodos de Aprendizagem Automática existentes na literatura com o objetivo de construir uma Híper-heurística para a seleção de Meta-heurísticas, na resolução do problema de escalonamento. Será realizado um estudo computacional com vista a analisar o desempenho da Híper-heurística proposta.

1.1 Objetivos e Principais Contribuições

O desenvolvimento de heurísticas que permitam resolver problemas de otimização combinatória (por exemplo, problemas de escalonamento) são confrontados com um significativo número de desafios. Um dos principais desafios é desenvolver um método aplicável a diferentes classes problemas, com diferentes tamanhos e propriedades, enquanto continuam a ser capazes de produzir soluções suficientemente boas num curto espaço de tempo. As Meta-heurísticas (Glover, 1986) (Glover e Kochenberger, 2003) (Talbi, 2009) e mais recentemente as Híper-heurísticas (Burke et al., 2003b) procuram resolver algumas destas questões.

Neste contexto, esta tese tem como objetivo o desenvolvimento de uma Híper-heurística, integrada no sistema Multiagente AutoDynAgents que incorpora aprendizagem baseada na experiência, para a resolução de novos problemas de escalonamento. Assim, identificam-se as principais contribuições desta dissertação:

- Revisão do estado da arte das abordagens de resolução de problemas de otimização combinatória, com enfâse no problema de escalonamento;
- Revisão do estado arte de técnicas de aprendizagem automática adequadas à aprendizagem por experiência e a sua adequação à resolução de problemas de escalonamento;
- Revisão do estado da arte sobre a abrangência das Híper-heurísticas como conjunto de abordagens motivadas (pelo menos em parte) pelo objetivo de automatizar o desenvolvimento de métodos (meta) heurísticos para resolver problemas que requerem um grande esforço computacional na pesquisa por uma boa solução;
- Proposta de uma Híper-heurística para integração no sistema AutoDynAgents para a resolução de novos problemas de escalonamento;

- Estudo computacional do sistema AutoDynAgents com a Híper-heurística, comparando-se com dados obtidos anteriormente e com outro método de seleção e afinação de Meta-heurísticas, o Raciocínio Baseado em Casos.

1.2 Estrutura do Documento

Nesta secção, será descrito de forma resumida cada um dos 7 capítulos que compõem esta dissertação.

No **primeiro capítulo** é realizado o enquadramento da tese, bem como a descrição dos objetivos e principais contribuições.

No **segundo capítulo** é abordado o problema do escalonamento e descritas algumas abordagens para a sua resolução, nomeadamente as Meta-heurísticas.

No **terceiro capítulo** é realizada uma revisão do estado da arte de técnicas de Aprendizagem Automática. Começa-se por descrever a Aprendizagem Automática e as suas subáreas, seguido de um enquadramento e descrição de diferentes técnicas, concluindo-se com uma sistematização das técnicas de aprendizagem automática.

No **quarto capítulo** é apresentado o termo Híper-heurística e algumas metodologias de classificação. Assim, começa-se pela introdução do conceito e formas de aplicabilidade, nomeadamente, na seleção de Meta-heurísticas para solucionar o problema de escalonamento. Neste contexto, foi realizada uma descrição mais detalhada sobre as Meta-heurísticas utilizadas. O capítulo é finalizado com a descrição da metodologia de seleção utilizada pela Híper-heurística, assim como a estratégia de aceitação das soluções obtidas.

No **quinto capítulo** é descrito o sistema AutoDynAgents e a Híper-heurística proposta e desenvolvida no âmbito desta tese, bem como a forma que esta se integra no sistema AutoDynAgentes. O capítulo é finalizado com um exemplo ilustrativo do funcionamento da Híper-heurística.

No **sexto capítulo** descreve-se o estudo computacional realizado com a Híper-heurística integrada no sistema AutoDynAgentes.

No **sétimo** e último **capítulo** são realizadas as conclusões, apresentadas as limitações, e trabalho futuro relacionado com o trabalho desenvolvido.

Capítulo 2

O Problema de Escalonamento

Para uma empresa, manter os lucros implica excelência na conversão das matérias-primas em produtos de valor, que corresponda às necessidades do cliente. Neste sentido, um plano de escalonamento, que traduza os planos de produção, de forma detalhada para o chão de fábrica, é um dos processos mais relevantes num sistema de produção. De facto, um plano de escalonamento otimizado apresenta grandes benefícios, como maior satisfação dos clientes, a diminuição dos níveis de *stock*, e o aumento da utilização dos recursos.

O problema de escalonamento consiste basicamente na realização de um conjunto de tarefas, num conjunto limitado de recursos, de forma a determinar a sua utilização, e assim satisfazer objetivos de carácter económico e operacional (Madureira, 2003). Segundo Brucker (2001), um problema de escalonamento é composto por três elementos: um conjunto de máquinas, as características específicas da tarefa, e os critérios de otimização. O conjunto de máquinas representa o tipo de sistema de produção que vai executar o plano de escalonamento. As características da tarefa representam o número de operações, as precedências entre as operações, e a possibilidade de interrupção. Por fim, o critério de otimização, onde o objetivo é a maximização ou minimização de uma função objetivo, como por exemplo, a minimização do tempo total de conclusão das tarefas ("*makespan*"), a minimização da soma dos atrasos pesados, o tempo médio do percurso, ou o atraso médio. Os três elementos mencionados especificam a variedade e complexidade de cada problema de escalonamento.

Neste capítulo serão apresentados alguns aspetos teóricos sobre o problema de escalonamento e descritas algumas abordagens de resolução. Estas abordagens dividem-se entre construtivas, por melhoramento e compostas, sendo dado mais enfâse a este último tipo, uma vez que se insere no âmbito deste trabalho.

2.1 Conceitos e Definições

Tradicionalmente, o problema de escalonamento assenta na otimização de problemas sujeitos a restrições. Baker (1974) refere a existência de quatro fases na abordagem aos problemas de escalonamento: formulação, análise, síntese e avaliação.

O problema de escalonamento apresenta uma natureza intrinsecamente dinâmica, onde os problemas reais de escalonamento apresentam restrições complexas e uma variedade de perturbações inesperadas. Na maioria dos ambientes de mundo-real, o escalonamento é um processo reativo progressivo onde a presença de informação em tempo real obriga continuamente à reconsideração e revisão dos planos pré-estabelecidos. A investigação em

escalonamento não tem considerado este problema, focando-se na otimização dos planos de escalonamento estático (Ouelhadj e Petrovic, 2008).

Genericamente, o escalonamento procura afetar no tempo, recursos designados de máquinas, a tarefas ("*jobs*"), sujeitas às restrições de que em qualquer instante nenhuma máquina processa mais de que uma tarefa, e nenhuma tarefa é processada simultaneamente, em mais do que uma máquina (Conway et al., 1967) (Baker, 1974) (Morton e Pentico, 1993) (Madureira, 2003).

Os métodos de escalonamento foram inicialmente propostos para a resolução de problemas industriais. No entanto, a sua aplicação pode ser considerada em áreas de prestação de serviços, como por exemplo, o tratamento de doentes num hospital, atendimento de clientes numa livraria, num supermercado, numa oficina de reparação de automóveis, ou no controlo de tráfego aéreo. Os recursos e tarefas apenas têm que assumir diferentes formas, conforme o ambiente em que se inserem. Assim, as máquinas representam pistas de aeroportos, quartos de hotéis, enfermeiros e salas em hospitais, professores e salas em escolas, unidades de processamento em computadores. Enquanto, as tarefas representam, os processos de fabrico de produtos, descolagens e aterragens, estadias em hotéis, tratamentos hospitalares, aulas e execução de programas em computadores. As tarefas caracterizam-se pelo tempo de processamento, data de entrega e a data de lançamento (Madureira, 2003).

2.2 Classificação do Problema de Escalonamento

De uma forma geral, o problema de escalonamento pode ser agrupado em duas categorias (Madureira, 2003): problema com tarefas uni-operação e problema com tarefas multi-operação.

O **problema com tarefas uni-operação** é constituído por uma única operação que é processada numa única máquina, em sistemas de máquina única ou sistemas de máquinas paralelas. Os sistemas de máquina única consistem no sequenciamento de um conjunto de tarefas, onde a ordenação das tarefas determina o plano de uma forma unívoca. Por outro lado, nos sistemas de máquinas paralelas, cada tarefa é processada numa única máquina de entre um conjunto de máquinas. Podem-se distinguir três tipos de modelos de máquinas paralelas (Madureira, 2003):

- **Máquinas idênticas**, onde a velocidade de processamento de qualquer das máquinas é igual, qualquer que seja a tarefa a processar;
- **Máquinas uniformes**, onde cada máquina tem a sua própria velocidade, geralmente diferente de outras máquinas;
- **Máquinas não relacionadas**, onde a velocidade de processamento de uma qualquer máquina depende da tarefa que está a ser processada, não estando relacionada com a velocidade das outras máquinas.

Os **problemas com tarefas multi-operação** ("*Flow-Shop*", "*Job-shop*" e "*Open-Shop*") são constituídos por um conjunto de operações a serem processadas em máquinas diferentes. No

Flow-Shop, múltiplos recursos organizados em série, refletem a sequência operatória comum a todas as tarefas. O ***Job-Shop*** consiste num conjunto de máquinas diferentes que processam as operações das tarefas. Cada tarefa é composta por um conjunto ordenado de operações que se caracterizam pela máquina onde são realizadas e pelo tempo de processamento. No ***Open-Shop***, um conjunto de tarefas é processado num número de máquinas diferentes, e onde a ordem de execução das operações de cada tarefa é irrelevante.

2.3 Abordagens de Resolução

Os métodos para a resolução de problemas de escalonamento, cujo tempo de resolução aumenta em função da dimensão dos problemas, podem ser divididos em duas categorias (Madureira, 2003): métodos exatos e métodos de aproximação.

Os métodos exatos realizam uma pesquisa exaustiva do espaço de soluções, garantindo assim a solução ótima. No entanto, devido a este facto, só devem ser considerados na resolução de problemas de pequenas dimensões. Estes métodos necessitam de bastante tempo para gerar uma solução ótima, e nem sempre conseguem produzir uma solução viável em tempo útil.

Por outro lado, os métodos de aproximação permitem encontrar soluções satisfatórios num período de tempo satisfatório. Fundamentados na Inteligência Artificial, estes métodos são de fácil implementação e produzem soluções em tempo útil. No entanto, não garantem a solução ótima.

Os métodos de aproximação podem ser divididos em três categorias: construtivos, por melhoramento e compostos. Os construtivos, constroem uma solução, passo a passo, segundo um conjunto de regras pré-estabelecidas, como por exemplo as Regras de Prioridade. Os métodos por melhoramento iniciam o processo a partir de uma qualquer solução viável, e procuram melhorá-la através de pequenas alterações, como por exemplo o algoritmo de Pesquisa Local. Os métodos compostos, primeiro têm uma fase construtiva e depois uma fase de melhoramento, onde se encontram as Meta-heurísticas e as Híper-heurísticas (Pereira, 2009).

2.3.1 Regras de Prioridade

Em escalonamento são frequentemente utilizados procedimentos simples para a determinação da sequência segundo a qual as operações devem ser realizadas, tendo em conta determinados critérios de desempenho (Madureira, 2003). Estes procedimentos são designados por Regras de Prioridade.

Neste contexto, apresenta-se de seguida algumas das Regras de Prioridade mais utilizadas (Baker, 1974) (Madureira, 2003):

- **EDD (*Earliest Due Date*)**: é dada prioridade à tarefa com menor data de entrega (*due date*), isto é, as tarefas são ordenadas por ordem crescente das suas datas de entrega;

- **SPT (*Shortest Processing Time*)**: é dada prioridade à tarefa com menor tempo de processamento na máquina em consideração;
- **LPT (*Longest Processing Time*)**: ao contrário da SPT, é dada prioridade à tarefa com maior tempo de processamento em determinada máquina;
- **FCFS (*First Come First Served*)**: a primeira tarefa a ser executada é aquela que estiver primeiramente disponível para processamento;
- **RND (*Random*)**: a ordem das tarefas é definida aleatoriamente.

As Regras de Prioridade apresentam-se como métodos de fácil implementação e que conseguem encontrar soluções razoáveis com alguma facilidade. No entanto, têm como desvantagem o facto de as soluções serem muitas vezes pobres devido à sua natureza míope (Ouelhadj e Petrovic, 2008).

2.3.2 Pesquisa Local

O algoritmo de Pesquisa Local não garante uma solução ótima. Essencialmente, a pesquisa local consiste na passagem de uma solução para outra, no domínio da sua vizinhança, de acordo com algumas regras bem definidas (Pirlot, 1996). A cada iteração, a vizinhança é avaliada e uma nova solução é selecionada. A solução candidata é aceite como próxima solução, desde que não seja pior que a solução atual, caso contrário a pesquisa termina.

2.3.3 Meta-Heurísticas

As Meta-heurísticas têm sido utilizadas com sucesso na resolução de problemas de escalonamento. As Meta-heurísticas são heurísticas de alto nível que guiam heurísticas de pesquisa local de forma a evitar os ótimos locais (Reeves, 1995) (Glover e Laguna, 1997) (Pham e Karaboga, 2000). Métodos, como *Simulated Annealing*, Pesquisa Tabu e Algoritmos Genéticos permitem melhorar os algoritmos de pesquisa local, quer seja pela aceitação de soluções piores, ou pela geração de soluções iniciais de uma forma mais inteligente do que simplesmente gerar soluções iniciais aleatórias (Ouelhadj e Petrovic, 2008). Assim, as Meta-heurísticas não garantem o ótimo global, no entanto, permitem obter soluções o mais próximo possível do ótimo global para um determinado problema.

É possível encontrar na literatura várias abordagens relativamente à abrangência das Meta-heurísticas, mas de forma genérica pode afirmar-se que são constituídas por Algoritmos Genéticos, Algoritmos Meméticos, Colónia Artificial de Abelhas, GRASP (*Greedy Randomized Adaptive Search Procedure*), Otimização por Colónia de Formigas, Pesquisa Tabu, *Particle Swarm Optimization*, *Simulated Annealing*, entre outras (Madureira, 2003) (Karaboga e Basturk, 2008) (Madureira, 2009). No âmbito deste trabalho, estas técnicas serão descritas com mais pormenor na secção 4.2.

2.3.4 Híper-heurísticas

O termo Híper-heurística foi introduzido por Cowling et al. (2000) e refere-se a um conjunto de algoritmos heurísticos que operam a alto nível de generalidade. São descritas como: "*uma heurística que gere o processo de seleção de qual a heurística ou método de nível mais baixo a ser aplicado num dado momento, dependendo das características das heurísticas e do espaço de soluções que está a ser explorado*" (Chakhlevitch e Cowling, 2008). O domínio de uma Híper-heurística é composto por conjuntos de (meta) heurísticas (e não pelo conjunto das instâncias de um problema de otimização). As Híper-heurísticas são o âmbito deste trabalho, como tal, o tema é abordado com detalhe no capítulo 4.

2.3.5 Outras Técnicas

Alguns sistemas de resolução de problemas de escalonamento adotaram técnicas de Inteligência Artificial tais como os Sistemas Baseados em Conhecimento, Redes Neuronais, Lógica Difusa, Raciocínio Baseado em Casos, etc. (Ouelhadj e Petrovic, 2008).

A motivação básica das abordagens baseadas em conhecimento prende-se com a existência de uma ampla variedade de conhecimento técnico especializado nas ações corretivas a tomar na presença de eventos de tempo-real. Os **Sistemas baseados em Conhecimento** focam-se em capturar o conhecimento especializado dos peritos num determinado domínio e usam um mecanismo de inferência para derivar conclusões ou recomendações em relação às ações corretivas a tomar. Para além disso, possuem o potencial para automatizar o raciocínio do perito humano e o conhecimento heurístico. No entanto, falta-lhes normalmente a capacidade para otimizar o sistema e requerem um esforço considerável no desenvolvimento e manutenção do sistema. Têm o objetivo de gerar soluções exequíveis de acordo com o domínio do problema, mas em termos de eficácia da capacidade na tomada de decisão, os Sistemas baseados em Conhecimento são limitados pela qualidade e integridade do conhecimento específico de domínio (Ouelhadj e Petrovic, 2008).

As **Redes Neuronais** procuram simular o cérebro humano, em particular, a sua capacidade de processamento de informação em paralelo (Osman e Kelly, 1996). Esta simulação é conseguida com auxílio a várias camadas de elementos de processamento simples, chamados de neurónios. Cada neurónio conecta-se aos seus vizinhos com diferentes coeficientes de conectividade. O processo de aprendizagem realiza-se através do ajuste destes coeficientes. As redes neuronais não garantem decisões ótimas, mas a sua capacidade de aprendizagem torna-as ideais para sistemas de mudanças rápidas (Ouelhadj e Petrovic, 2008).

A **Lógica Difusa** foi proposta por Zadeh (1965), onde o objetivo consistia em tratar dados vagos ou imprecisos. Assim, o autor introduziu o conceito de conjunto *fuzzy*, de forma a tratar matematicamente valores linguísticos imprecisos, como por exemplo "alto" ou "baixo". O conjunto *fuzzy* são funções que mapeiam um valor, que pode ou não pertencer ao conjunto, compreendido entre zero e um, indicando assim o seu grau de pertença. A Lógica Difusa procura

identificar que afirmações lógicas sobre a realidade não são absolutamente verdadeiras ou falsas, mas que têm um certo grau de verdade.

O **Raciocínio baseado em Casos** é uma abordagem de resolução de problemas que reutiliza o conhecimento adquirido a partir de tentativas passadas, de forma a resolver um problema futuro. Cada tentativa passada de resolução de um problema é armazenada como um novo caso. A coleção de casos, na base de casos, torna-se então o modelo. Quando o sistema resolve um problema, em vez de iniciar o processo do zero, procura na base de casos os registos cujos atributos são similares. Assim, consegue apresentar uma solução com base no grau de similaridade com o problema. A precisão da base de casos aumenta com o número de casos armazenados (Pereira, 2009) (Pereira e Madureira, 2013).

2.4 Sumário do Capítulo

Neste capítulo foi descrito o problema de escalonamento, bem como as duas categorias de classificação do problema, nomeadamente, problema com tarefas uni-operação e problema com tarefas multi-operação, onde neste ultimo encontram-se os problemas do tipo *Flow-Shop*, *Job-Shop* e *Open-Shop*.

Foram abordadas algumas estratégias de resolução do problema de Escalonamento, sendo descritos os métodos de aproximação construtivos (onde se inclui as Regras de Prioridade), por melhoramento, e compostos (Meta-heurísticas e Híper-heurísticas).

Capítulo 3

Aprendizagem Automática

A Inteligência Artificial (IA), nos últimos anos, passou por um extraordinário crescimento. As suas ideias e métodos foram amplamente utilizados em diferentes áreas. Entre as aplicações de maior sucesso, encontra-se o desenvolvimento de sistemas periciais, a implementação prática de sistemas de compreensão linguística, os avanços significativos nos sistemas de visão e reconhecimento de voz, a locomoção robótica, entre outros. Como disciplina da área do conhecimento, a IA foi altamente enriquecida por áreas como a Filosofia, a Matemática, a Psicologia, as Ciências dos Computadores, a Engenharia e as Ciências Cognitivas.

A Inteligência Artificial é uma área que procura desenvolver métodos computacionais que simulem a inteligência humana. Um dos requisitos básicos de um qualquer comportamento inteligente é a aprendizagem. Como tal, não pode existir inteligência sem processo de aprendizagem. Neste contexto, a aprendizagem automática ("*machine learning*") assume-se como uma das áreas da investigação em IA que se dedica ao desenvolvimento de algoritmos e técnicas que permitam dotar os computadores com a capacidade de aprendizagem. Mitchell (2006) definiu a questão da seguinte forma: *"Como construir sistemas computacionais que aprendem automaticamente com a experiência e quais as leis fundamentais que governam todo o processo de aprendizagem"*.

A aprendizagem automática estuda as leis fundamentais que governam todo o processo de aprendizagem. Há muito que as teorias e algoritmos desenvolvidos por esta área do conhecimento são de enorme relevância para a compreensão dos aspetos da aprendizagem humana. A cognição humana revela um enorme potencial para a pesquisa da aprendizagem automática, dado que o ser humano tem uma capacidade inata para adquirir novo conhecimento, desenvolver capacidades motoras através da instrução ou prática, organizar novo conhecimento e descobrir novos factos por observação e experimentação (Carbonell et al., 1983). Compreender o processo de aprendizagem humana tem sido e continua a ser um dos grandes objetivos da IA. O estudo e a modelação computacional dos processos de aprendizagem nas suas múltiplas manifestações constituem o objetivo da área de aprendizagem automática.

3.1 Perspetiva Histórica

No decurso dos anos, a investigação na área de aprendizagem automática evolui utilizando diferentes abordagens e dando ênfase a diferentes aspetos e objetivos.

Neste contexto, três grandes paradigmas emergiram (Carbonell et al., 1983):

- **Redes neuronais e técnicas de decisão teórica**;
- **Aprendizagem simbólica**;
- **Sistemas de aprendizagem do conhecimento**.

O primeiro paradigma procura desenvolver sistemas de aprendizagem em que a estrutura inicial é reduzida ou inexistente. O grande impulso na pesquisa desta abordagem envolveu a construção de uma variedade de máquinas baseadas no modelo neuronal, com uma estrutura inicial aleatória ou parcialmente aleatória. Este sistema, de uma forma geral, é referido como redes neuronais. O método de aprendizagem em tais sistemas consistia em mudanças incrementais das probabilidades que elementos do tipo neurónios transmitiriam sobre a forma de sinais (Carbonell et al., 1983).

Os recursos tecnológicos eram escassos, pelo que, muita da investigação realizada sobre este paradigma era teórica ou envolvia a construção de *hardware* experimental, tal como os *perceptrons* (Rosenblatt, 1958). O fundamento que serviu de suporte a este paradigma foi estabelecido nos anos quarenta por McCulloch e Pitts (1943), que descobriram a aplicabilidade da lógica simbólica para modelar a atividade do sistema nervoso e por Rashevsky (1948) e pela sua equipa na área de biofísica matemática (Carbonell et al., 1983).

A experiência adquirida pelo vasto esforço de investigação nesta área originou o aparecimento de uma nova disciplina reconhecimento de padrões, levando ao desenvolvimento de uma abordagem de decisão teórica para a aprendizagem automática. No entanto, os resultados práticos procurados pelas abordagens de modelação neuronal e decisão teórica tiveram um sucesso limitado, pois nunca alcançaram o resultado desejado e a investigação nesta área entrou em declínio. Estudos teóricos revelaram fortes limitações dos sistemas de aprendizagem do tipo *perceptron* (Minsky e Papert, 1969).

Na década de 60 a aprendizagem simbólica começou a emergir, resultado de trabalhos realizados nas áreas da Psicologia e Inteligência Artificial. Este paradigma utilizava lógica ou representações de estruturas gráficas em vez de métodos numéricos ou estatísticos. Os sistemas aprendiam descrições simbólicas que representavam conhecimento de alto nível e efetuavam suposições estruturais sobre os conceitos a adquirir (Carbonell et al., 1983).

No início da década de 70 os investigadores foram para além dos conceitos isolados de aprendizagem e começaram a investigar um vasto espectro de métodos de aprendizagem, baseados em sistemas do conhecimento. Este paradigma pode ser caracterizado por algumas tendências, incluindo (Carbonell et al., 1983):

- **Abordagens baseadas em conhecimento:** O uso de conhecimento orientado à tarefa e as restrições de tal processo na aprendizagem. Os anteriores sistemas apresentavam limitações no processo de aprendizagem, isto é, para adquirir novo conhecimento, o sistema já deveria possuir uma enorme quantidade de conhecimento inicial;

- **Exploração de métodos alternativos de aprendizagem**: Além das anteriores pesquisas em aprendizagem por processos, passou a haver uma maior variedade de métodos de aprendizagem, tal como aprender por instruções, aprender por analogia e descoberta de conceitos e classificação;
- **Incorporar a capacidade de gerar e selecionar tarefas de aprendizagem**: Em contraste com esforços anteriores, alguns sistemas utilizavam heurísticas para controlar o foco de atenção, através da geração de tarefas de aprendizagem, propondo experiências de forma a reunir dados de treino e escolhendo os conceitos a adquirir.

A década de 80 revelou-se um ponto de viragem para a área científica de aprendizagem automática. Várias descobertas foram registadas, tendo-se verificado o aparecimento de diversas novas técnicas de aprendizagem. O conceito de *inductive bias* foi apresentado por Utgoff e Mitchell (1982). Este conceito define os pressupostos que devem ser adicionados aos dados observados para transformar as saídas do algoritmo de aprendizagem automática em deduções logicas. Durante esta época, o algoritmo de *BackPropagation* (inicialmente descrito por Werbos (1974)) foi redescoberto por Rumelhart et al. (1986). Este algoritmo permitiu ultrapassar muitas das limitações apresentadas por Minsky e Papert (1969) sobre as redes neuronais.

No início da década de 90 a Aprendizagem Automática voltou a ganhar nova popularidade com a intersecção do conhecimento entre as Ciências dos Computadores e a Estatística. Estas sinergias resultaram num novo modo de pensar na área da Inteligência Artificial: a abordagem probabilística. Neste tipo de abordagem a incerteza dos parâmetros passaram a ser incorporadas nos modelos. A área passou a seguir uma abordagem orientada aos dados, por oposição aos sistemas de conhecimento desenvolvidos até então. Muitos dos sucessos da Aprendizagem Automática resultaram das ideias que foram sendo apresentadas e desenvolvidas durante esta década.

O estudo da Aprendizagem Automática desenvolveu-se devido ao esforço que a comunidade científica colocou na descoberta de métodos que permitissem dotar os computadores com a capacidade de aprendizagem e assim imitar o cérebro humano. Além disso, a contínua evolução dos algoritmos de aprendizagem tornou possível o desenvolvimento de sistemas com carácter comercial. De facto, o número de domínios que recebem o contributo da área da Aprendizagem Automática é já significativo, nomeadamente, em sistemas de reconhecimento de voz, robótica, sistemas de visão, ciências empíricas, entre outros (Mitchell, 2006).

3.2 Tipos de Aprendizagem Automática

Os algoritmos de aprendizagem automática estão organizados segundo uma determinada taxonomia, em função do resultado desejado. Genericamente, os algoritmos e técnicas de aprendizagem automática dividem-se em três tipos (Alonso et al., 2001):

- **Aprendizagem Supervisionada**: o algoritmo gera uma função que realiza o mapeamento entre a entrada e a saída dos dados. Ou seja, estamos perante um problema de classificação, onde o aprendiz é obrigado a aprender uma função que deve conseguir prever o valor de saída correto para qualquer objeto de entrada válida;
- **Aprendizagem não Supervisionada**: modela um conjunto de dados de entrada: exemplos classificados não se encontram disponíveis;
- **Aprendizagem por Reforço**: o algoritmo aprende a política de como agir em função da observação do mundo. Todas as ações realizadas têm impacto no ambiente, e o ambiente dá feedback de forma a orientar o algoritmo de aprendizagem.

3.2.1 Aprendizagem Supervisionada

A Aprendizagem Supervisionada procura definir uma função, que consiga a partir de um conjunto de dados de treino realizar uma previsão sobre novos dados. Estes dados de treino consistem em pares de objetos de entrada-saída. A função pode gerar um valor contínuo (regressão) ou pode prever a classe do novo objeto (classificação). O objetivo do algoritmo supervisionado é prever o valor da função para cada objeto de entrada após analisar um conjunto de exemplos de treino.

No entanto, este tipo de aprendizagem apresenta algumas limitações. A primeira prende-se com a dificuldade na classificação dos dados. Quando existe uma elevada quantidade de dados de entrada, é extremamente dispendioso, se não impossível, classificar todos os dados. A segunda, com o facto de que nem tudo no mundo real é passível de ser classificado, existem incertezas e ambiguidades. Estas dificuldades podem limitar os sistemas de aprendizagem em alguns cenários. Assim, um sistema de aprendizagem, após analisar um determinado número de exemplos, normalmente pequeno, deve fornecer um classificador que funcione bem com qualquer exemplo.

A Aprendizagem Supervisionada é uma técnica utilizada, normalmente no treino de Redes Neuronais, Árvores de Decisão, *Support Vector Machines*, Aprendizagem Baseada em Instâncias, Aprendizagem de Regras, Redes Bayesianas, entre outras.

3.2.2 Aprendizagem não Supervisionada

A Aprendizagem não Supervisionada procura determinar a forma como os dados se encontram organizados. De acordo com Ghahramani (2008), este tipo de algoritmos são desenvolvidos para extrair estruturas dos dados. Assim, o objetivo não é maximizar uma função de utilidade, mas simplesmente encontrar semelhanças nos dados de treino. De facto, uma das técnicas da Aprendizagem não Supervisionada é o Agrupamento de Dados (*Clustering*), onde o objetivo consiste em agrupar objetos que apresentem um grau de similaridade entre si.

As técnicas de agrupamento de dados revelam-se particularmente úteis na descoberta de distribuições com significado ou classes em dados. O agrupamento de dados consiste na divisão de um conjunto de dados em grupos, de forma a colocar os diferentes objetos de dados em grupos com a mesma similaridade. Como tal, a análise de agrupamento de dados não é uma tarefa automática, mas um processo iterativo de descoberta de conhecimento que envolve um processo de tentativa e erro, onde muitas vezes é necessário modificar os parâmetros e o pré-processamento até que resultado desejado seja alcançado.

3.2.3 Aprendizagem por Reforço

A Aprendizagem por Reforço é uma área que procura analisar as decisões de um aprendiz num determinado ambiente de forma a maximizar a noção de recompensa cumulativa. Sutton e Barto (1998) definiram a Aprendizagem por Reforço como a aprendizagem que permite o mapeamento entres as situações e as ações de forma a maximizar um sinal numérico de recompensa. Na Aprendizagem por Reforço não é indicado ao aprendiz o tipo de ação que deve realizar, mas este deve descobrir, por experimentação, qual a ação que irá permitir obter uma maior recompensa. Este tipo de aprendizagem distingue-se da Aprendizagem Supervisionada pelo facto de nunca ter pares de dados de entrada-saída corretos, nem ações sub-ótimas explicitamente corretas. A Aprendizagem por Reforço é particularmente útil em domínios onde a informação de reforço (expressa como penalizações ou recompensas) é fornecida após serem realizadas sequências de ações no ambiente (Panait e Luke, 2005).

Assim, o processo de aprendizagem dá-se pela interação com o ambiente. A cada iteração um aprendiz observa o estado corrente s. Em cada estado o aprendiz pode realizar uma ação a partir do conjunto de ações disponíveis. Uma ação pode provocar uma transição de um estado s para um estado s_0, com base numa probabilidade de transição. Um valor numérico de recompensa r é devolvido ao aprendiz para informar sobre a "qualidade" das suas ações.

Existem dois tipos diferentes de métodos de Aprendizagem por Reforço, nomeadamente, política de iteração e valor de iteração. Um aprendiz que procura diretamente pela melhor política num espaço de possíveis políticas está a aplicar um método de política de iteração. Por outro lado, os métodos de valor de iteração não procuram diretamente pela política ideal, em vez disso aprendem funções de avaliação para os estados ou pares estado-ação. As funções de avaliação, como o algoritmo *Q-Learning*, podem ser utilizadas para avaliarem a qualidade de um par estado-ação.

3.3 Técnicas de Aprendizagem Automática

Nesta secção, serão descritas algumas técnicas de Aprendizagem Automática, nomeadamente Árvores de Decisão, Aprendizagem Baseada em Instâncias, Aprendizagem de Regras, Redes Neuronais Artificiais, Redes Bayesianas, *Support Vector Machines*, Agrupamento de Dados e *Q-Learning*.

3.3.1 Árvores de Decisão (*Decision Trees*)

Uma árvore de decisão é uma estrutura de dados hierárquica que implementa uma estratégia de dividir para conquistar. É um método não paramétrico eficiente, que pode ser utilizado na classificação como na regressão (Alpaydin, 2004). Como indicado na Figura 1, a estrutura hierárquica da árvore de decisão encontra-se dividida em três partes: nó raiz (previsão do tempo), nós internos (vento e humidade) e nós folha (sim ou não). Cada nó de decisão m implementa uma função de teste $f_m(x)$. A função executa o teste a um atributo da instância e dependendo do resultado um dos ramos é escolhido. Este processo inicia-se no nó raiz e é repetido até alcançar o nó folha. O valor da folha representa o resultado do problema.

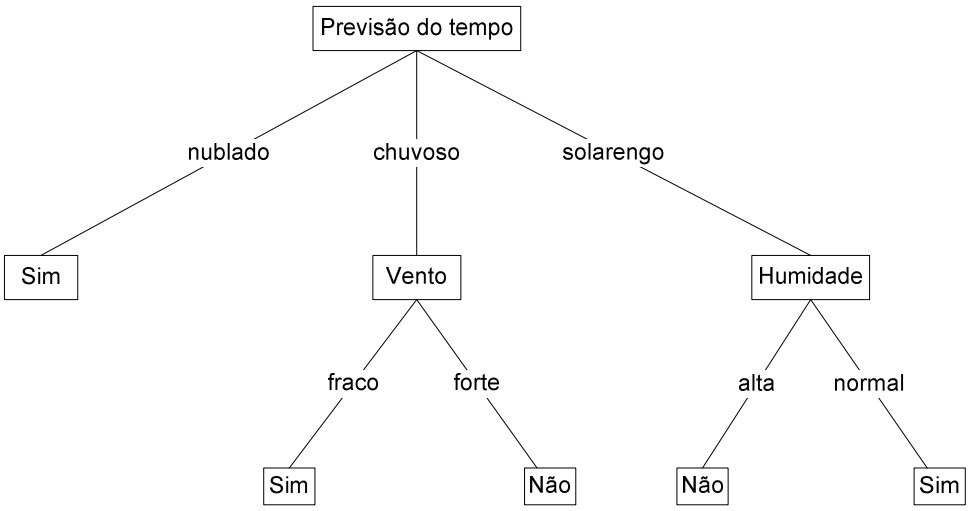

Figura 1 – Representação de uma Árvore de Decisão (adaptado (Mitchell, 1997))

A função de teste $f_m(x)$ define um discriminante no espaço de entrada d-dimensional dividindo-o em regiões mais pequenas, regiões estas que são subdivididas conforme o processo se desloca no sentido descendente da árvore. Diferentes algoritmos de árvores de decisão assumem diferentes modelos para a função. Cada nó folha tem um rótulo de saída. No caso da classificação é o código da classe e no caso da regressão é um valor numérico. Um nó folha define uma região localizada no espaço de entrada onde as instâncias que "caem" nessa região têm o mesmo resultado de saída. Os limites das regiões são definidos pelos discriminantes que são codificados nos nós internos que se encontram no caminho entre o nó raiz e o nó folha.

3.3.2 Aprendizagem Baseada em Instâncias (*Instance Based Learning*)

Os algoritmos de aprendizagem baseada em instâncias derivam do classificador vizinho mais próximo (Cover e Hart, 1967). Para além disso, apresentam semelhanças às versões revistas dos algoritmos do vizinho mais próximo (Hart, 1968) (Gates, 1972) (Dasarathy, 1980), que também guardam e utilizam apenas as instâncias selecionadas para gerar previsões de classificação.

Em contraste com outros algoritmos de aprendizagem (árvores de decisão, redes neuronais, redes Bayesianas, entre outros) que constroem uma descrição genérica e explícita da função objetivo a partir dos exemplos de treino, os algoritmos de aprendizagem baseada em instâncias armazenam simplesmente os exemplos de treino. Assim, cada vez que uma instância é recebida, a função objetivo é processada com base no conhecimento disponibilizado pelos exemplos de treino anteriormente armazenados. Os algoritmos de aprendizagem baseada em instâncias assumem que as instâncias podem ser representadas como pontos num espaço euclidiano. Para além disso, estes algoritmos também podem utilizar representações simbólicas mais complexas para as instâncias. Os algoritmos baseados em instâncias são muitas vezes denominados de algoritmos de aprendizagem lenta (*lazy-learning*), pois o processo de generalização é atrasado até que seja necessário classificar uma nova instância. A principal vantagem deste tipo de aprendizagem lenta traduz-se na capacidade de estimar a função objetivo localmente e de forma diferente para cada nova instância a ser classificada (Mitchell, 1997).

Um dos algoritmos mais simples baseado em instâncias é o algoritmo K-vizinhos-mais-próximos (K-*Nearest Neighbours* – K-NN). Este algoritmo baseia-se no princípio de que as instâncias que fazem parte de um conjunto de dados, geralmente existem na proximidade de outras instâncias que têm propriedades muito semelhantes. Se as instâncias estão classificadas, então uma instância não classificada pode ser determinada por observação da classe vizinha mais próxima. O algoritmo localiza a instância K mais próxima da instância que se pretende consultar e determina a sua classe por identificação da classe que contém a classificação mais frequente (Kotsiantis, 2007).

3.3.3 Aprendizagem de Regras (*Rules Learning*)

As árvores de decisão podem ser traduzidas num conjunto de regras onde é criada uma regra para cada caminho entre o nó raiz e uma das folhas da árvore. No entanto, as regras também podem ser inferidas a partir dos dados de treino, utilizando uma variedade de algoritmos baseados em regras.

A classificação de regras representa cada classe na forma de expressões disjuntivas. Uma equação disjuntiva apresenta a seguinte forma: $(X_1 \wedge X_2 \wedge \ldots \wedge X_n) \vee (X_{n+1} \wedge X_{n+2} \wedge \ldots \wedge X_{2n}) \vee \ldots \vee (X_{(k-1)n+1} \wedge X_{(k-1)n+2} \wedge \ldots \wedge X_{kn})$, onde K representa o numero de disjunções, n é o numero de conjunções em cada disjunção, e X_n é definido pelo alfabeto $X_1, X_2, \ldots, X_j \cup \sim X_1, \sim X_2, \ldots, \sim X_j$. O objectivo é construir um conjunto de regras que seja pequeno e consistente com os dados de treino. Normalmente, a aprendizagem de um grande número de regras traduz-se na memorização do conjunto de dados de treino, o que não é desejável.

Um dos algoritmos baseado em regras mais conhecido é o RIPPER. Este algoritmo cria regras através de duas fases: crescimento repetido (*growing*) e *pruning*[1]. Na fase de crescimento repetido as regras são mais restritivas de modo a que os dados de treino estejam o mais

[1] Esta técnica tem como objetivo reduzir a complexidade do classificador bem como melhorar a precisão da previsão pela remoção de secções do classificador que podem ser baseadas em dados com ruído ou errados.

próximo possível e na fase de *pruning* as regras são menos restritivas de modo a evitar uma excessiva aproximação dos dados de treino, o que poderia causar um mau desempenho em instâncias não observadas (Kotsiantis, 2007).

A principal característica dos algoritmos baseados em regras é a sua compreensibilidade. Além disso, a precisão da classificação efetuada pelos algoritmos de aprendizagem de regras pode ser melhorada pela combinação de características (tal como nas árvores de decisão), como a utilização do conhecimento do utilizador ou através de algoritmos automáticos de construção de características.

3.3.4 Redes Neuronais Artificiais (*Artificial Neural Networks*)

As Redes Neuronais Artificiais têm como base o trabalho realizado por McCulloch e Pitts (1943). Este trabalho originou a criação de um modelo computacional para as redes neuronais, que permitiu a evolução da investigação nesta área.

As Redes Neuronais Artificiais são um modelo computacional inspirado em padrões biológicos e procedimentos que simulam a atividade cerebral humana. A estrutura de redes neuronais mais utilizada para a classificação de dados é a rede *perceptron* multicamada, ou MLP (*Multi Layer Perceptron*). Estas redes foram criadas para resolver as limitações apresentadas pelas redes *perceptron* de uma camada (*Single Layer Perceptron*). Uma rede *perceptron* multicamada é constituída por um grande número de unidades (neurónios), onde juntos formam uma rede (Figura 2).

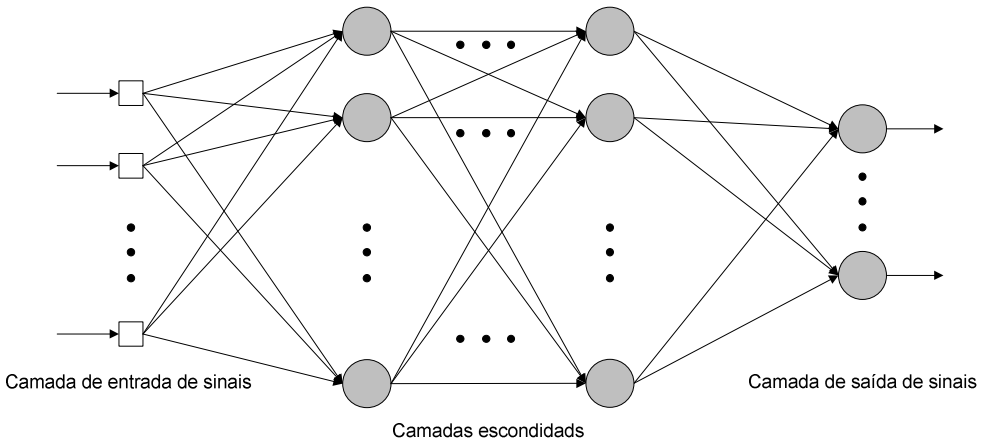

Figura 2 – Representação de uma Rede Neuronal

As unidades presentes numa rede neuronal normalmente têm uma de três categorias (Kotsiantis, 2007): unidades de entrada de dados, que recebem a informação a ser processada; unidades de saída de dados, onde são encontrados os resultados do processo; e unidades que se encontram nas camadas intermédias, também conhecidas por unidades ocultas ou escondidas.

Durante a classificação, o sinal que se encontra nas unidades de entrada de dados é propagado por toda a rede para determinar os valores de ativação das unidades de saída de dados. Cada unidade de entrada de dados tem um valor de ativação que representa uma característica externa à rede. Como tal, cada unidade de entrada de dados envia o seu valor de ativação às unidades intermédias que se encontram diretamente conectadas. Cada uma das unidades intermédias procede ao cálculo do seu próprio valor de ativação, e envia-o às unidades de saída de dados (Kotsiantis, 2007). O valor da ativação que cada unidade recebe é calculado através de uma função de ativação. A função executa o somatório dos sinais produzidos pelo produto entre os pesos das conexões e as entradas fornecidas à unidade. A saída da unidade é definida pelo seu valor de ativação calculado da seguinte forma (equação (1)):

$$v_j = \sum_{i=1}^{m} w_{ji} x_i + b \qquad (1)$$

onde:

- v é o valor de ativação da unidade j;
- w são os pesos das ligações da unidade j;
- x é o valor de cada um dos m estímulos que chegam à unidade j;
- b é o valor do *bias*[2].

O algoritmo de aprendizagem *BackPropagation* é o mais conhecido e utilizado nas Redes Neuronais Artificiais para estimar o valor dos pesos.

3.3.5 Redes Bayesianas (*Bayesian Networks*)

O termo "Redes Bayesianas" foi inventado por Judea Pearl em (1985). No entanto, só no final dos anos 80 é que Judea Pearl (1988) e Richard E. Neapolitan (1989) resumiram as propriedades desta técnica e instituíram as Redes Bayesianas como campo de estudo.

Uma Rede Bayesiana, também conhecida como rede de crença, rede probabilística ou rede causal, é um modelo que utiliza teoria dos grafos, condições de Markov e distribuição de probabilidades para representar conhecimento sobre um domínio desconhecido. Esta representação é efetuada através de variáveis e estados, onde a partir destes é possível realizar inferências. A estrutura de uma Rede Bayesiana é um grafo acíclico onde as variáveis são os nós e os arcos as relações entre as variáveis (Figura 3):

- Um conjunto de variáveis e um conjunto de arcos que efetuam a ligação entre as variáveis;
- Cada variável possui um número limitado de estados mutuamente exclusivos;
- As variáveis e os arcos formam um grafo acíclico dirigido;

[2] O termo *bias* é utilizado para representar o erro sistemático ou tendenciosidade.

- Para cada variável X_i que possui como pais $X_1, \ldots, X_n$ existe uma tabela de probabilidade condicional $P(X_i \mid X_1 \wedge \ldots \wedge X_n)$.

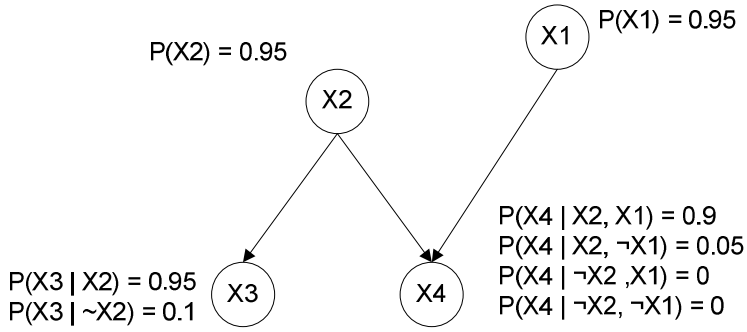

Figura 3 – Estrutura de uma Rede Bayesiana (adaptado de (Kotsiantis, 2007))

Tipicamente, o processo de aprendizagem de uma Rede Bayesiana pode ser dividido em duas fases (Kotsiantis, 2007): aprendizagem da estrutura do grafo acíclico da rede e a determinação dos seus parâmetros. Os parâmetros probabilísticos são codificados num conjunto de tabelas, uma para cada variável, na forma de distribuições condicionais de uma variável. Dadas as independências codificadas na rede, a distribuição conjunta pode ser reestruturada através da multiplicação destas tabelas.

As Redes Bayesianas podem ser representadas, numa forma simplificada, pelas Redes Naive Bayes, onde estas são compostas por grafos acíclicos dirigidos com apenas um nó pai (representando o nó não observado) e vários nós filhos (correspondendo a nós observados). Os filhos neste tipo de rede apresentam, no contexto dos pais, um elevado grau de independência entre si (Good, 1950).

3.3.1 Support Vector Machines

O algoritmo de *Support Vector Machines* (SVM) foi originalmente desenvolvido por Vladimir N. Vapnik, e a versão atual proposta por Corinna Cortes e Vapnik em 1993 e publicado em Cortes e Vapnik (1995)

O funcionamento base das SVM incorpora o princípio de Minimização do Risco Estrutural[3] (*Structural Risk Minimisation* – SRM), que tem mostrado ser superior ao tradicional princípio de Minimização do Risco Empírico[4] (*Empirical Risk Minimisation* – ERM), utilizado pelas redes neuronais convencionais. A SRM minimiza o limite superior da dimensão VC (erro de generalização), em oposição à ERM que minimiza o erro nos dados de treino. É esta a diferença que equipa as SVM com uma grande capacidade de generalização. As SVM foram desenvolvidas

[3] É um princípio indutivo utilizado na aprendizagem automática. Este princípio descreve um modelo genérico de controlo da capacidade e fornece um *tradeoff* entre a complexidade do espaço de hipóteses e a qualidade do ajuste dos dados de treino (erro empírico).

[4] É um princípio teórico da aprendizagem estatística que define uma família de algoritmos de aprendizagem e é utilizado para fornecer limites teóricos no desempenho dos algoritmos de aprendizagem.

para resolver problemas de classificação, mas também permitem a resolução de problemas de regressão (Gunn, 1998).

O problema da classificação pode ser limitado à consideração do problema de duas classes sem perda de generalidade. Neste problema o objetivo é separar as duas classes com o auxílio de uma função, sendo esta inferida a partir dos exemplos disponíveis. O objetivo é produzir um classificador que funcione adequadamente com exemplos nunca observados, isto é, que generalize de forma adequada. Considere-se o exemplo apresentado na Figura 4, onde existem vários classificadores lineares capazes de separar os dados, mas existe apenas um capaz de maximizar a margem – representa cada um dos lados de um híper-plano que separa duas classes de dados. Este classificador linear é denominado de híper-plano de separação ótimo (Gunn, 1998).

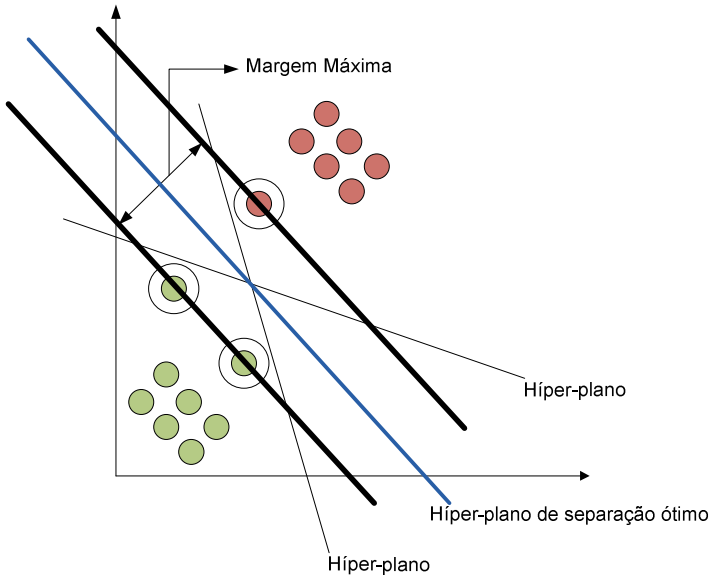

Figura 4 – Híper-plano de separação ótimo (adaptado de (Kotsiantis, 2007))

As SVM também podem ser aplicadas aos problemas de regressão através da introdução de uma função alternativa de perda. A função de perda (*loss function*) tem que ser modificada de modo a incluir a métrica de distância. Em Gunn (1998) o autor apresenta quatro possíveis funções de perda: a função de perda quadrática, a função de perda Laplaciana, a função de perda proposta por Huber e a função de perda ε-insensível.

3.3.2 Agrupamento de dados (*Clustering*)

As abordagens de agrupamentos de dados podem ser divididas em cinco categorias: abordagens por partição, hierárquicas, baseadas em densidade, baseadas em grelha e baseadas em modelos. Das cinco categorias apresentadas duas são amplamente utilizadas (Jain, 2010): hierárquica e partição. Os algoritmos de agrupamento de dados hierárquico realizam o agrupamento de dados recursivamente, onde o processo pode ser efetuado de forma

aglomerada ou divisiva. No primeiro, o processo é iniciado com os pontos representativos dos dados de um agrupamento e vai sucessivamente efetuando a reunião dos pares de agrupamentos similares até formar uma hierarquia de agrupamentos. No segundo, o processo é iniciado com os pontos representativos dos dados de um agrupamento e recursivamente vai dividindo cada agrupamento em agrupamentos mais pequenos. Por outro lado, os algoritmos de agrupamento de dados por partição encontram todos os agrupamentos simultaneamente como uma partição dos dados e não impõem uma estrutura hierárquica. A entrada de dados num algoritmo hierárquico é representada por uma matriz do tipo $n \times n$, onde n é o número de objetos a serem agrupados. Em contrapartida, o algoritmo por partição pode utilizar uma matriz do tipo $n \times d$, onde n objetos estão incorporados num espaço d-dimensional, ou uma matriz do tipo $n \times n$ (Duarte, 2008) (Jain, 2010).

Entre os vários algoritmos representativos das duas categorias apresentadas, existem três que se destacam. Os algoritmos hierárquicos *single-link* e *complete-link* e o algoritmo por partição k-médias (k-means) (Jain, 2010).

O **algoritmo *single-link*** define a distância entre agrupamentos como sendo a menor distância entre todos os possíveis pares de elementos de dois agrupamentos. Podendo ser considerado como um grafo completamente conectado, com os nós a corresponderem às instâncias e as ligações entre nós com pesos iguais às distâncias entre as instâncias. Neste contexto, o algoritmo *single-link* corresponde à construção de uma árvore de distâncias mínimas para este grafo. Por outro lado, o **algoritmo *complete-link*** define a distância entre dois agrupamentos como sendo a maior distância entre todos os pares possíveis.

O **algoritmo K-médias** recebe como parâmetros, k, o número de grupos pretendido e inicialmente escolhe de forma aleatória k objetos de um conjunto de dados X como os centros (centróides) iniciais de cada grupo, $\{\bar{x}_1, \dots, \bar{x}_k\}$. Após este primeiro passo, o algoritmo k-médias limita-se a atribuir cada objeto x_i ao grupo C_k cujo centro $\bar{x}_k$ se encontra mais próximo e em seguida atualizar o centro de grupo, $\bar{x}_k$, como sendo o vetor médio dos $|C_k|$ objetos associados ao grupo, $\bar{x}_k = \frac{\sum x_{i \in C_k} x_i}{|C_k|}$. Este processo repete-se até que não exista qualquer modificação nos grupos de uma iteração para a seguinte, garantindo que a função-objetivo converge para um ótimo (local).

3.3.3 Q-Learning

O algoritmo *Q-Learning* foi proposto por Watkins em (1989). O objetivo do algoritmo é aprender o valor do par estado-ação, $Q(s, a)$, que a longo prazo representa a recompensa esperada para cada par estado-ação. Os valores Q aprendidos com este algoritmo tendem a convergir para os valores de estado-ação ótimos (Watkins e Dayan, 1992).

Estes valores são calculados segundo a seguinte equação (2):

$$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \, max \, Q(s', a') - Q(s, a) \right] \qquad (2)$$

onde, s representa o estado anterior, s' o novo estado, a a ação realizada no estado s, a' a possível ação no estado s', r a recompensa recebida, $\alpha \in [0,1]$ a taxa de aprendizagem, e $\gamma \in [0,1]$ a taxa de desconto que indica a importância de recompensas futuras. Os valores representam a política ótima que o aprendiz tem a intenção de aprender. A política é uma regra que o aprendiz utiliza na seleção das ações. Quando o valor estado-ação é aprendido, a política ótima pode ser construída, bastando selecionar a ação com o valor mais alto em cada estado. Um dos pontos fortes do *Q-learning* é a sua capacidade de comparar a utilidade esperada das ações disponíveis sem a necessidade de um modelo do ambiente (Panait e Luke, 2005).

O algoritmo *Q-Learning* é conhecido como um método de aprendizagem de diferença temporal, pois utiliza a diferença entre a nova e a antiga estimativa da função valor.

3.4 Avaliação das Técnicas de Aprendizagem Automática

Um sistema de aprendizagem apresenta como principal vantagem a capacidade de adaptação a um ambiente em mudança. As técnicas de aprendizagem automática ainda estão longe de dotar os computadores com capacidade de aprendizagem igual à dos seres humanos. No entanto, os algoritmos desenvolvidos são eficazes na aprendizagem de determinadas tarefas. Os algoritmos de aprendizagem automática têm demonstrado grande valor prático, em particular:

- Problemas de *data mining* relacionados com grandes bases de dados, onde é possível encontrarem regularidades implícitas nos dados de forma automática (Fawcett e Provost, 1996) (Tseng e Hwang, 2007) (Moro et al., 2011);
- Em domínios onde o ser humano não tem conhecimento suficiente para desenvolver algoritmos eficientes (por exemplo, aprender a jogar ou aprender a interpretar expressões faciais em função das emoções) (Niese et al., 2011) (Chou et al., 2011);
- Em domínios em que o programa tem que se adaptar a ambientes dinâmicos (Laddaga, 2002) (Wang et al., 2005) (Nallur e Bahsoon, 2013).

No entanto, grande parte dos algoritmos de aprendizagem automática requer uma fase de treino, especialmente na fase de extração da informação (generalização do conhecimento), o que torna a adaptação *online* (aprendizagem sustentada) difícil. Assim, grande parte das técnicas de aprendizagem automática apresentam dificuldades de aprendizagem em ambientes dinâmicos. Mais ainda, as técnicas de aprendizagem automática são *data oriented*, ou seja, modelam as relações contidas nos dados de treino. Como tal, se o conjunto de dados de treino não é uma seleção representativa do domínio do problema, o modelo resultante pode ser diferente do domínio do problema real. Esta limitação, torna-se ainda mais acentuada pelo facto de que a maioria das técnicas de aprendizagem automática não permite conhecimento *a priori*. Além disso, os algoritmos de aprendizagem automática apresentam dificuldades em lidar

com o ruído. Muitas técnicas conseguem evitar o ruído, no entanto, ao faze-lo correm o risco de ignorar características importantes do domínio do problema.

3.4 Sumário do Capítulo

Neste capítulo foi apresentada uma visão geral do campo da Aprendizagem Automática, bem como os tipos de aprendizagem, sobretudo a Aprendizagem Supervisionada, a Aprendizagem Não-Supervisionada, e a Aprendizagem por Reforço. A Aprendizagem Automática dedica-se ao desenvolvimento de algoritmos e técnicas que permitam dotar os computadores com a capacidade de aprendizagem.

Neste contexto, foi realizada a descrição de algumas técnicas de Aprendizagem Automática, nomeadamente Árvores de Decisão, Aprendizagem Baseada em Instâncias, Aprendizagem de Regras, Redes Neuronais Artificiais, Redes Bayesianas, *Support Vector Machines*, Agrupamento de Dados e *Q-Learning*. Para além disso, também foi realizada uma avaliação sucinta das técnicas de Aprendizagem Automática.

Capítulo 4

Meta-Heurísticas e Híper-Heurísticas

Apesar do sucesso das Meta-heurísticas na resolução de problemas que requerem um grande esforço computacional, onde se inclui o problema de escalonamento, a sua aplicação na resolução de novas instâncias apresenta algumas dificuldades. Estas dificuldades surgem, principalmente, devido à quantidade de parâmetros e à forma como os algoritmos são selecionados. Além disso, o desempenho das diferentes Meta-heurísticas pode variar significativamente dependendo das características específicas do problema em análise. Neste contexto, o objetivo é desenvolver algoritmos que sejam de aplicação mais genérica. Assim, com a utilização das Híper-heurísticas, procura-se encontrar a Meta-heurísticas adequada para solucionar um determinado problema, em vez de tentar resolver o problema diretamente. A Híper-heurística pode ser vista como uma metodologia (de alto nível) que, quando aplicada procura resolver de forma eficaz um determinado problema ou classe de problemas.

O termo Híper-heurística foi introduzido em 1997 para descrever um protocolo que combina vários métodos de IA. O termo foi utilizado de forma independente em 2000 para descrever "*heurísticas que escolhem heurísticas*" (Cowling et al., 2000) no contexto da otimização combinatória. Neste contexto, uma Híper-heurística é uma abordagem de alto nível, onde dada uma determinada instância de um problema e um número de heurísticas de baixo-nível, escolhe e aplica a heurística mais apropriada a cada momento de decisão. No entanto, a ideia de automatizar o processo de modelação de heurísticas não é nova, uma vez que esta remonta ao início da década de 60 (Fisher e Thompson, 1961, 1963). A investigação mais recente na área das Híper-heurísticas procura automatizar a geração de novas heurísticas mais adequadas para um dado problema ou classe de problemas. Isto é tipicamente feito através da combinação de componentes das heurísticas (Burke et al., 2013).

A literatura apresenta uma grande variedade de abordagens de Híper-heurísticas que utilizam metodologias de alto-nível, juntamente com um conjunto de heurísticas de baixo nível, aplicadas a diferentes problemas de otimização. Neste contexto, Chakhlevitch e Cowling (2008) apresentam três critérios que permitem definir estas abordagens: uma Híper-heurística é (i) uma heurística de alto nível que gere um conjunto de heurísticas de baixo nível, (ii) procura um bom método para resolver um problema, em vez de uma boa solução, e (iii) utiliza apenas informação específica do problema em análise. Os autores consideram o último critério como sendo o mais relevante. Em Burke et al. (2009), a Híper-heurística é definida, como "*um método de procura ou mecanismo de aprendizagem utilizado na seleção ou geração de heurísticas para resolver problemas computacionais de pesquisa*", onde uma classificação genérica é proposta: (i) a natureza do espaço de pesquisa da Híper-heurística, (ii) e a origem do *feedback* durante o processo de aprendizagem. Esta classificação é demonstrada na Figura 5.

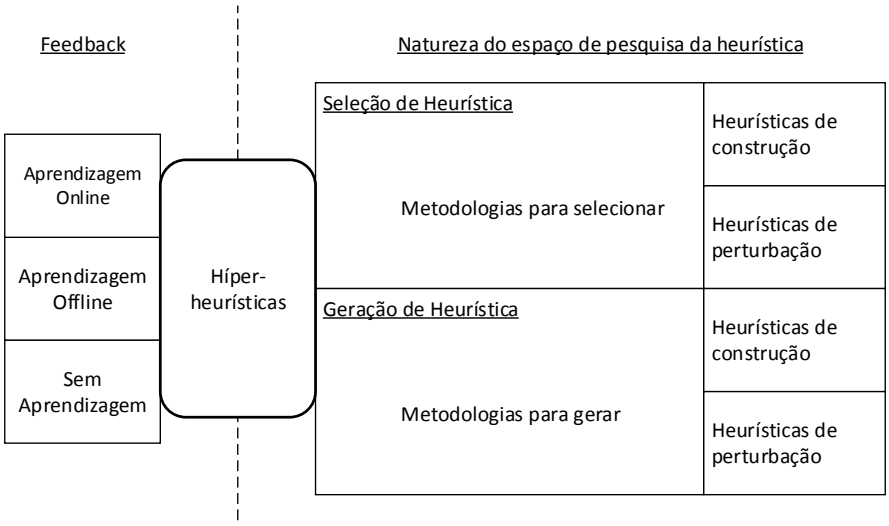

Figura 5 – Classificação das Híper-heurísticas (adaptado de (Burke et al., 2013))

Nesta classificação, os diferentes espaços de pesquisa heurísticos podem ser combinados com diferentes fontes de *feedback*, e com diferentes técnicas de aprendizagem automática. Assim, a classificação proposta e apresentada na Figura 5 pode ser sumariada da seguinte forma (Burke et al., 2013):

- Relativamente à natureza do espaço de pesquisa da Híper-heurística
    - Metodologia de seleção de heurísticas: Produzir combinações pré-existentes:
        - Heurísticas de construção;
        - Heurísticas de perturbação.
    - Metodologia de geração de heurísticas: Gerar novos métodos heurísticos que utilizem componentes básicos (blocos de construção):
        - Heurísticas de construção;
        - Heurísticas de perturbação.
- Relativamente à origem do *feedback* durante o processo de aprendizagem
    - Híper-heurísticas de aprendizagem *online*: aprender, enquanto uma instância de um problema é resolvida, por exemplo, o uso de reforço para a seleção de heurística e, geralmente o uso de meta-heurísticas como estratégias de pesquisa de alto nível;
    - Híper-heurísticas de aprendizagem *offline*: aprender, a partir de um conjunto de instâncias de treino, por exemplo, Raciocino Baseado em Casos e programação genética;
    - Híper-heurísticas sem aprendizagem: não utilizam o feedback do processo de pesquisa.

4.1 Perspetiva Histórica

A ideia que suporta as Híper-heurísticas não é nova. De facto, desde do início da década de 1960 que esta ideia emergiu na comunidade científica, mais precisamente em áreas como as ciências da computação e a inteligência artificial.

Na década de 60, Fisher e Thompson (1963) e Crowston et al. (1963) apresentaram a hipótese em que a combinação de regras de escalonamento em sistemas de produção fabril produziu resultados superiores à sua utilização individual. Este trabalho, descrito como pioneiro, e muito à frente do seu tempo, propôs um método para combinar regras de escalonamento, através da utilização da aprendizagem probabilística. Note-se, que no início da década de 60, as meta-heurísticas e os métodos de pesquisa local encontravam-se ainda em desenvolvimento. No entanto, a abordagem de aprendizagem proposta assemelhava-se a um algoritmo de pesquisa local estocástico.

Na década de 90, estas ideias foram revistas, e Storer et al. (1992b) afirmaram que o principal problema consistia em criar uma boa combinação de heurísticas específicas para determinado problema de pesquisa, e definir uma vizinhança dentro do espaço de pesquisa de uma heurística. Assim, através da perturbação das combinações heurísticas, dos dados do problema, ou ambos, foi encontrado um subconjunto de novas soluções. Estas vizinhanças tornaram-se a base para os métodos de pesquisa local. Este princípio, devido à sua natureza genérica, pode ser aplicado às heurísticas para resolver um amplo espectro de problemas. Uma vez que, a enfâse é colocada no desenvolvimento de espaços de pesquisa, os autores utilizaram uma variante do método de pesquisa local. No entanto, outros métodos poderiam ser utilizados, como, o *Simulated Annealing*, os Algoritmos Genéticos ou a Pesquisa Tabu. Estas ideias foram implementadas e testadas, através do desenvolvimento de heurísticas para resolver o problema de escalonamento *Job-Shop*.

Fang et al. (1993) utilizou um algoritmo genético para escolher uma heurística num espaço de procura de heurísticas no contexto do escalonamento *Open-Shop*. A abordagem produziu bons resultados sobre problemas de referência, incluindo soluções já conhecidas naquele tempo. Hart e Ross (1998) aplicaram com sucesso uma variante desta ideia ao problema de escalonamento *Job-Shop* dinâmico. A abordagem baseou-se na evolução de algoritmos heurísticos anteriormente propostos para a escolha de tarefas de escalonamento. Foram obtidos bons resultados em problemas de referência quando testados com diferentes critérios. Hart et al. (1998), também utilizou um algoritmo genético para resolver um problema de escalonamento do mundo real.

Norenkov e Goodman (1997), também utilizaram algoritmos evolucionários para procurar num espaço de heurísticas. Os autores consideraram a utilização de problemas de escalonamento *Flow-Shop*, e incluir dois tipos de regras de escalonamento: (i) para determinar a ordenação das tarefas, e (ii) para determinar as tarefas que são afetadas as máquinas. Foram realizados diferentes conjuntos de experiências e a qualidade das soluções dependeu fortemente do subconjunto de heurísticas aplicadas.

Note-se que as abordagens mencionadas e discutidas anteriormente são *online*. Isto é, direcionadas para encontrar boas sequências de heurísticas para resolver uma determinada instância de um problema. Além disso, o problema estudado foi em todos os casos relacionado com escalonamento *Job-Shop*. No entanto, a comunidade de IA também contribui para o conceito de Híper-heurística. Especialmente, através do trabalho realizado em sistemas de planeamento automático. Em (Gratch e Chien, 1996) (Gratch et al., 1993), o sistema COMPOSER foi utilizado para controlar o horário de comunicações via satélite, envolvendo um número de satélites em órbita da Terra e três estações terrestres. O sistema utiliza o método de pesquisa local que procura num espaço de possíveis estratégias de controlo. O sistema de aprendizagem propõe alternadamente alterações à estratégia de controlo atual, e avalia estatisticamente se a alteração produziu melhores resultados. A abordagem é *offline*, no sentido em que, é necessária uma grande quantidade de problemas de treino, de forma a ser possível obter uma estimativa da utilidade esperada para as várias estratégias de controlo.

4.2 Meta-Heurísticas

Dada a sua importância, existe uma procura contínua por melhores soluções para os problemas de escalonamento. Os modelos, para tais problemas são cada vez mais complexos e a pesquisa exaustiva por soluções ótimas é geralmente impraticável, devido ao tempo computacional requerido. Portanto, são utilizados métodos heurísticos que não garantem soluções ótimas. Pois o seu principal objetivo é produzir soluções de qualidade aceitável em tempo razoável. No entanto, nem sempre o seu desempenho é o melhor, devido à variedade de instâncias de problemas. Existe, um vasto número de heurísticas, conhecidas na literatura, que são especificamente desenvolvidas e ajustadas para resolver certas classes de problemas de otimização combinatória. Estes métodos baseiam-se na pesquisa parcial do espaço de soluções e são conhecidos por Meta-heurísticas.

As Meta-heurísticas são algoritmos, que se têm revelado muito eficazes na resolução de vários problemas de otimização combinatória, a sua aplicação é geralmente limitada a determinados domínios de problemas. Para além disso, as meta-heurísticas incorporam informação específica de um problema e requerem extenso conhecimento, sobre o domínio do problema, e sobre a técnica heurística apropriada (Chakhlevitch e Cowling, 2008). Estes métodos, normalmente, são de desenvolvimento dispendioso. As abordagens Meta-heurísticas, com um bom desempenho num determinado problema do mundo real, podem não funcionar ou até mesmo produzir más soluções para outros problemas ou instâncias da mesma classe. Tais limitações podem tornar-se problemáticas em situações onde os dados do problema e os requisitos do negócio mudam com frequência ao longo do tempo.

Assim, as Meta-Heurísticas apresentam-se como métodos iterativos capazes de obter soluções o mais próximo possível do ótimo global para um determinado problema. O caminho percorrido pelo método, e assumindo que as soluções se encontram relacionadas entre si, ou seja, a partir de cada solução é possível alcançar o conjunto de soluções para o problema, chama-se de trajetória no espaço de soluções.

Neste contexto, descrevem-se de seguida algumas Meta-Heurísticas, dando enfâse às utilizadas no âmbito deste trabalho, nomeadamente, os Algoritmos Genéticos, a Colónia Artificial de Abelhas, a Otimização por Colónia de Formigas, o *Particle Swarm Optimization*, a Pesquisa Tabu e o *Simulated Annealing*.

4.2.1 Algoritmos Genéticos

Os Algoritmos Genéticos constituem o grupo mais extenso dos métodos representativos da aplicação das ferramentas de Computação Evolucionária. Estes trabalham com uma população de indivíduos que representam as soluções para o problema, usando operadores de recombinação, mutação, seleção e substituição (Madureira, 2003).

O primeiro Algoritmo Genético proposto na literatura foi desenvolvido por John Holland (1975). O algoritmo apresentado na Tabela 1, genericamente é iniciado com uma população de N indivíduos (soluções) e evolui produzindo uma população do mesmo tamanho em cada iteração. A geração $(n+1)$ de soluções é obtida a partir da geração de ordem n, $X^{(n)}$ através do procedimento ilustrado na Figura 6, onde os melhores indivíduos (soluções) de $X^{(n)}$ são selecionados e cruzados entre si, produzindo uma descendência de soluções que substitui os "maus" indivíduos da população atual. A mutação é geralmente realizada numa pequena porção de descendentes. Cada solução da população atual $\{x_1, \dots, x_n\}$ é avaliada pelo seu enquadramento (*"fitness"*), o qual pode ser simplesmente o valor da função objetivo num problema de maximização ou minimização. Cada cromossoma[5] representa uma solução num determinado espaço de soluções (Madureira, 2003).

Tabela 1 – Algoritmo Genético

Algoritmo Genético
1. Inicialização da população de indivíduos
2. Avaliação do enquadramento de cada individuo da população
3. **Repetir**
4. Escolha dos melhores indivíduos para reprodução
5. Geração de novos indivíduos através de operações de cruzamento e mutação
6. Avaliação das soluções e seleção do melhor indivíduo
7. Substituição dos "maus" indivíduos e criação da nova geração
8. **Até** critério de paragem satisfeito
9. **Devolver** melhor solução encontrada

[5] O termo cromossoma será utilizado para designar um indivíduo da população, pelo que ambos os termos serão utilizados indistintamente. Na natureza, frequentemente, os indivíduos são constituídos por vários cromossomas (por exemplo, o ser humano tem 46 cromossomas).

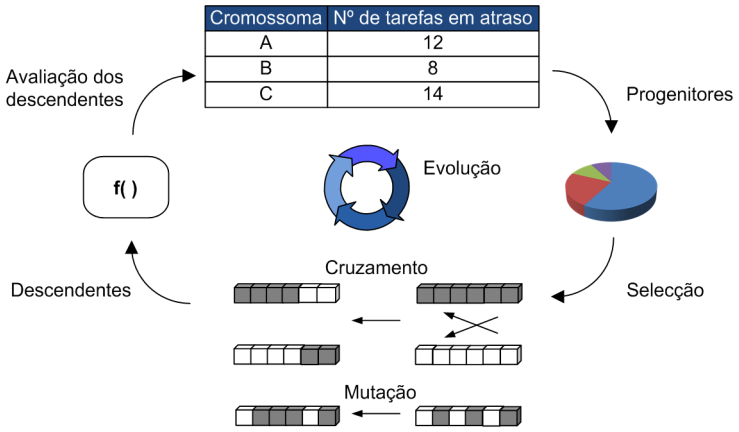

Figura 6 – Ciclo de uma geração de um Algoritmo Genético (adaptado de (Madureira, 2003))

A seleção dos melhores indivíduos de uma dada população é realizada de acordo com o seu enquadramento. Os pares de indivíduos selecionados são então submetidos, com determinada probabilidade T_c, à operação de cruzamento. Cada descendente é por sua vez submetido à mutação com determinada probabilidade T_m. O passo final na geração de uma nova população é a substituição dos "maus" indivíduos da população atual pelos descendentes, possivelmente mutados. Este procedimento produz a geração $(n+1)$. O algoritmo termina geralmente depois de um número de gerações predefinido ter sido criado (Madureira, 2003).

É importante que seja mantida uma diversidade suficiente na população, de forma a permitir a exploração de muitas regiões do espaço das soluções, e não restringir a pesquisa à vizinhança do máximo local (Madureira, 2003).

Os Algoritmos Genéticos foram aplicados na resolução de problemas de escalonamento em, por exemplo (Fang et al., 1993) (Lee et al., 1997) (Branke, 2000) (Madureira, 2003) (Madureira et al., 2007) (Madureira et al., 2009).

4.2.2 Colónia Artificial de Abelhas

A Colónia Artificial de Abelhas (*Artificial Bee Colony* – ABC) é um algoritmo de otimização baseado no comportamento das colónias de abelhas, proposto por Karaboga (2005). O algoritmo é inspirado no comportamento real das abelhas. No modelo ABC, a colónia é constituída por três grupos de abelhas: as trabalhadoras, as exploradoras e as espectadoras. Para cada fonte de alimento existe apenas uma abelha trabalhadora. Por outras palavras, o número de abelhas trabalhadoras na colónia é igual ao número de fontes de alimentos existentes. Quando uma abelha trabalhadora regressa à colmeia, realiza uma determinada dança, onde, através dos movimentos, transmite a informação sobre a proximidade e a quantidade de néctar da fonte de alimento. As abelhas expectadoras assistem à dança e escolhem a fonte de alimento. No entanto, se uma abelha trabalhadora abandona a fonte de alimentação, passa a ser uma abelha exploradora e começa a procurar de novas fontes de alimento.

O ABC é um algoritmo baseado na pesquisa populacional. O algoritmo realiza uma pesquisa da vizinhança combinado com uma pesquisa aleatória. Esta pesquisa aleatória, apesar de não garantir, torna possível encontrar o ótimo global e evitar ficar preso em ótimos locais.

Tabela 2 – Algoritmo Colónia Artificial de Abelhas

Algoritmo Colónia Artificial de Abelhas
1. Inicialização da população de abelhas
2. Avaliação do *fitness* da população de abelhas
3. Ciclo = 1
4. **Repetir**
5. Fase das abelhas trabalhadoras
6. Calcular as probabilidades para as abelhas exploradoras
7. Fase das abelhas espectadoras
8. Fase das abelhas exploradoras
9. Memorizar a melhor solução alcançada até ao momento
10. Ciclo = ciclo +1
11. **Até** ciclo = Número Máximo de Ciclos
12. **Devolver** melhor solução encontrada

O algoritmo apresentado na Tabela 2 é uma versão modificada para problemas de otimização combinatória, proposto por (Karaboga e Akay, 2011). No algoritmo a colónia é constituída por metade de abelhas trabalhadoras e a outra metade por abelhas espectadoras. Como já mencionado, se uma abelha trabalhadora abandonar a fonte de alimento, torna-se numa abelha exploradora. Quando aplicado a um problema, as soluções são representadas pelas fontes de alimento, onde a qualidade da fonte de alimento ou solução corresponde à quantidade de néctar. Quanto maior a quantidade de néctar, melhor a solução encontrada.

O algoritmo inicia o processo com a inicialização da população, seguido de um processo de avaliação. O próximo passo é a fase das abelhas trabalhadoras, onde é atribuída às abelhas a responsabilidade pelas fontes de alimentos já descobertos. Para além disso, também transmitem à colónia a informação sobre a qualidade da fonte de alimento pela qual são responsáveis. Esta troca de informação com as abelhas espectadoras é realizada na área de dança, que se traduz no cálculo de um conjunto de probabilidades. De seguida, na fase das abelhas espectadoras, é selecionada de forma probabilística e proporcional ao seu valor de *fitness* uma solução. A ideia é que uma abelha espectadora avalie a informação apresentada pelas abelhas trabalhadoras na área de dança e escolha uma fonte de alimento com uma probabilidade relacionada com a quantidade de néctar. Finalmente, na fase das abelhas exploradoras, se a exploração de uma determinada fonte de alimento deixa de ser viável, esta é suprimida e substituída por uma nova. A nova fonte de alimento é descoberta por uma abelha exploradora.

Aplicações desta Meta-heurística à resolução de problemas de escalonamento podem ser encontradas em, por exemplo (Chong e Low, 2006) (Pham et al., 2007) (Karaboga e Akay, 2011) (Fera et al., 2013).

4.2.3 Otimização por Colónia de Formigas

A Meta-Heurística Otimização por Colónia de Formigas (*Ant Colony Optimization* – ACO) baseia-se no comportamento real das formigas, onde este permite encontrar o menor caminho entre a fonte de alimento e a respetiva colónia (Blum e Roli, 2003) (Dorigo e Stützle, 2004) (Madureira, 2009). Enquanto percorrem o caminho, as formigas depositam uma substância chamada feromona e, quando escolhem um caminho, optam, com maior probabilidade, por aquele que possui maior quantidade de feromona, pois será o mais curto, e aquele percorrido pelo maior número de formigas. Assim, os algoritmos de Otimização por Colónia de Formigas são baseados em modelos probabilísticos parametrizados, que utiliza feromonas para a definição de um caminho.

Tabela 3 – Algoritmo Otimização por Colónia de Formigas (adaptado de (Talbi, 2009))

Algoritmo Otimização por Colónia de Formigas
1. Inicializar os caminhos de feromonas
2. **Repetir**
3. **Para** cada formiga **Fazer**
4. Construir a solução a partir do caminho de feromonas
5. Atualizar o caminho de feromonas
6. Evaporação da feromona
7. Reforço da feromona
8. **Fim Para**
9. **Até** critério de paragem satisfeito
10. **Devolver** a melhor solução encontrada ou conjunto de soluções

O primeiro algoritmo ACO (Tabela 3) proposto na literatura foi designado de *Ant System* (AS) (Dorigo et al., 1991). O algoritmo começa pela inicialização do valor da feromona, de seguida, a cada nova iteração do algoritmo cada formiga constrói uma solução com base na feromona presente. Estas soluções são então utilizadas para atualizar os valores da feromona.

Na fase de construção, a formiga constrói incrementalmente a solução, adicionando componentes para a solução já existente. A escolha probabilística do próximo componente da solução a ser adicionado é realizada através de probabilidades de transição, que são determinadas pela equação (3):

$$P(C_r|S_a[C_l]) = \begin{cases} \text{Se } C_r \in J(S_a[C_l]): \dfrac{[\eta_r]^{\alpha}[\tau_r]^{\beta}}{\sum_{C_u \in J\,(S_a)[C_l]}[\eta_r]^{\alpha}[\tau_r]^{\beta}} \\ \text{Senão: } 0 \end{cases} \qquad (3)$$

onde α e β são parâmetros para ajustar a importância da informação heurística (η_r) e os valores de feromona (τ_r), respectivamente. J$(S_a[C_l])$ refere-se ao conjunto de componentes da solução $S_a[C_l]$ onde C_l é o último componente adicionado (Blum e Roli, 2003).

Após todas as formigas terem construído uma solução, é realizada a atualização da feromona, através de uma regra definida pela equação (4):

$$\tau_j \leftarrow (1-\rho) * \tau_j + \sum_{a \in A} \Delta\tau_j^{S_a}$$
$$\forall\, T_j \in T, \text{onde } \Delta\tau_j^{S_a} = \begin{cases} \text{Se é componente de } S_a: F(S_a) \\ \text{Senão: } 0 \end{cases} \quad (4)$$

Esta regra visa aumentar o valor da feromona dos componentes que encontrarem a melhor solução. Além disso, $(1-\rho) * \tau_j$ define a taxa de evaporação da feromona, de modo a que os maus caminhos se degradem ao longo do tempo. De notar, que o incremento de feromona é realizado apenas nos caminhos percorridos pela formigas.

O critério de paragem pode ser determinado por um número máximo de iterações, ou então pelo fenómeno da estagnação, ou seja, quando todas as formigas percorrem o mesmo caminho.

A Otimização por Colónia de Formigas foi aplicada na resolução de problemas de escalonamento em, por exemplo (Zwaan et al., 1999) (Haipeng et al., 2004) (Ventresca e Ombuki, 2004) (Yoshikawa e Terai, 2006) (Madureira et al., 2012).

4.2.4 Particle Swarm Optimization

O *Particle Swarm Optimization* é um método baseado em populações proposto e desenvolvido por James Kennedy e Russell Eberhart (1995), que procura simular um sistema social de uma forma simplificada.

Este método, procura demonstrar o comportamento que bandos de pássaros ou cardumes de peixes assumem nas suas trajetórias locais aleatórias, mas globalmente determinadas. Os bandos de pássaros ou cardumes de peixes efetuam movimentos coordenados e sincronizados como forma de descobrir comida ou como mecanismo de autodefesa (Madureira, 2009).

Duma forma computacional, estes algoritmos surgem como uma abstração desse comportamento biológico natural onde a procura por uma melhor posição é a busca por uma solução ótima, sendo o conjunto de posições da partícula o espaço de busca ou espaço de soluções possíveis. O comportamento de cada partícula baseia-se na sua experiência anterior e na das outras partículas com que se relaciona (Madureira, 2009). Esta Meta-heurística salvaguarda as melhores posições encontradas, que teoricamente significam a melhor solução encontrada.

Tabela 4 – Algoritmo *Particle Swarm Optimization* (adaptado de (Talbi, 2009))

Algoritmo Particle Swarm Optimization
1. Inicialização aleatória de todo o *swarm*
2. **Repetir**
3. Avaliar $f(x_i)$
4. **Para** todas a partículas i **Fazer**
5. Atualizar as velocidades
6. Mover para a nova posição
7. **Se** $f(x_i) < f(pbest_i)$ então $pbest_i = x_i$ **Fim Se**
8. **Se** $f(x_i) < f(gbest_i)$ então $gbest_i = x_i$ **Fim Se**
9. Atualizar x_i, v_i
10. **Fim Para**
11. **Até** critério de paragem satisfeito
12. **Devolver** melhor solução

Este método é um pouco diferente de outras técnicas evolucionárias. Existem N partículas, e cada uma dessas partículas ajusta a sua direção com base na sua própria experiência, e na experiência da restante população (grupo de partículas). Estas partículas encontram-se inseridas no espaço de soluções, e fundamentam-se em procedimentos determinísticos para fazerem a pesquisa do ótimo local. Cada movimento de otimização de cada partícula é baseado em três parâmetros, o fator de sociabilidade, o fator de individualidade, e a velocidade máxima (Kennedy e Eberhart, 1995):

- **Fator social (C1):** determina a atracão (convergência) das partículas para a melhor solução descoberta por um elemento do grupo;
- **Fator cognitivo (C2):** determina a atracão da partícula com a sua melhor posição encontrada;
- **Fator velocidade:** delimita o movimento, uma vez que esse é direcional e determinado.

O algoritmo apresentado na Tabela 4 começa pela inicialização da posição atual e velocidades de todas as partículas. A cada iteração, e enquanto o critério de paragem não for atingido, é calculado o valor de aptidão de cada partícula, sendo atualizado o melhor valor local de cada uma (*pBest*). Após, este processo, é atualizado o melhor valor global (*gBest*) e são calculadas as novas velocidades e posições de todas as partículas. No final, a melhor solução é devolvida de acordo com *gBest*.

Considerando-se o conjunto das posições atuais de cada partícula como $x_i = \{x_{i1}, x_{i2}, \dots, x_{id}\}$, onde x representa a posição da partícula i na dimensão d, a posição atual de cada partícula é atualizada com base na equação (5):

$$x_{id} = x_{id} + v_{id} \tag{5}$$

e a velocidade atualizada com base na equação (6):

$$v_i(t) = W \times v_i(t-1) + C_1 \times \big(p_i - x_i(t-1)\big) + C_2 \times (p_g - x_i(t-1)) \qquad (6)$$

O peso da inércia W é usado para calcular o impacto da velocidade anterior na nova velocidade, determinando as capacidades para as partículas explorarem zonas locais ou globais. Se a inércia for elevada, as partículas terão capacidade para exploração global, enquanto se o seu valor for mais baixo, a pesquisa tenderá a ser centrada e mais refinada. Sendo assim, o valor para W deve ser um valor intermédio, que permita às partículas ter a capacidade de exploração global e local apropriadas para encontrar uma solução satisfatória com menos custos (Kennedy e Eberhart, 1995).

Esta Meta-heurística foi aplicada a problemas de escalonamento em, por exemplo (Liu et al., 2006), (Sha e Hsu, 2006) (Pongchairerks e Kachitvichyanukul, 2009) (Yen e Ivers, 2009).

4.2.5 Pesquisa Tabu

A Pesquisa Tabu, desenvolvida por Fred W. Glover (1986), é um método baseado em pesquisa local que têm como objetivo escapar aos mínimos locais. Esta estratégia de pesquisa utiliza estruturas de memória que descrevem as soluções visitadas. Assim, se uma possível solução foi anteriormente visitada, é marcada como "tabu", de modo a que o algoritmo não considere a solução de forma repetida.

Esta Meta-heurística, como descrita na Tabela 5, inicia o processo a partir de uma solução inicial e escolhe para solução seguinte a melhor solução na vizinhança atual, ou de uma sub-vizinhança. De notar, que a escolha da nova solução acontece, independentemente de a solução ser pior que a solução atual.

Tabela 5 – Algoritmo Pesquisa Tabu (adaptado de (Talbi, 2009))

Algoritmo Pesquisa Tabu
1. $S = S_0$, solução inicial
2. Inicialização da lista tabu, memória a médio e longo prazo
3. **Repetir**
4. Encontrar melhor vizinho admissível, que não seja tabu
5. $S = S'$
6. Atualizar lista tabu, condições de aceitação, memória a médio e longo prazo
7. **Se** critério de intensificação então intensificação **Fim Se**
8. **Se** critério de diversificação então diversificação **Fim Se**
9. **Até** critério de paragem satisfeito
10. **Devolver** melhor solução encontrada

A Pesquisa Tabu, utiliza procedimentos iterativos de procura local em vizinhanças para se mover de uma solução x para uma potencial solução melhorada x' na vizinhança de x, até que um qualquer critério de paragem seja satisfeito. Os algoritmos de pesquisa local, com alguma regularidade ficam bloqueados em soluções de má qualidade. De forma a garantir que tal não aconteça, o método de Pesquisa Tabu explora cuidadosamente a vizinhança de cada solução. As soluções aceites na nova vizinhança, $N^*(x)$, são determinadas através do uso de estruturas de memoria. Estrutura esta que se divide em memória a médio (intensificação) e longo (diversificação) prazo. Utilizando estas estruturas de memória, a procura avança de forma iterativa, movendo-se da solução corrente x para uma melhor solução x' em $N^*(x)$.

Estas estruturas de memória formam as listas tabu, que se traduzem, num conjunto de regras e soluções proibidas utilizadas para filtrar as soluções a serem aceites na vizinhança $N^*(x)$. Na sua forma mais simples, uma lista tabu representa o conjunto das soluções que foram visitadas no passado.

Aplicações desta Meta-heurística à resolução de problemas de escalonamento podem ser encontradas em, por exemplo (Dell'Amico e Trubian, 1993) (Mehta e Uzsoy, 1999) (Ponnambalam et al., 2000) (Madureira, 2003) (Madureira et al., 2007) (Madureira et al., 2009).

4.2.6 Simulated Annealing

O *Simulated Annealing* é um algoritmo com origem no anos 80, proposto por Kirkpatrick et al. (1983) e Cerny (1985), com ligação à termodinâmica e à metalurgia (Metropolis et al., 1953), cuja motivação original surgiu com base no processo de arrefecimento do metal em fusão, com tendência para solidificar numa estrutura de energia mínima. Este algoritmo assenta numa base estatística, e baseia-se em movimentos resultantes de soluções de má qualidade para uma solução atual de forma a escapar ao mínimo local. Este processo acontece através da combinação de uma geração aleatória e de um melhoramento iterativo a partir de comparações e probabilidades.

De forma análoga, o algoritmo *Simulated Annealing* substitui a solução atual por uma solução próxima, isto é, na vizinhança do seu espaço de soluções, escolhida de acordo com uma função objetivo e a variável de temperatura. Quanto maior for a temperatura, maior a componente aleatória introduzida na próxima solução. À medida que o algoritmo evolui, o valor da temperatura vai sendo diminuído gradualmente, tornando o algoritmo mais seletivo na aceitação de novas soluções, convergindo assim para uma solução ótima.

O algoritmo apresentado na Tabela 6 começa por gerar uma solução inicial S, que pode ser gerada aleatoriamente ou construída através duma heurística, e inicializa a temperatura T. Enquanto o critério de paragem não for atingido, para cada iteração é selecionada aleatoriamente uma solução S' pertencente à vizinhança de S. Esta solução vizinha S' é avaliada para ser aceite como nova solução, o que acontece se for melhor do que S. Neste caso, a melhor solução global também é atualizada se necessário. Caso S' seja pior do que S, então é aceite com base numa probabilidade.

Tabela 6 - Algoritmo *Simulated Annealing* (adaptado de (Talbi, 2009))

Algoritmo Simulated Annealing
1. $S = S_0$, geração da solução inicial
2. $T = T_{max}$, temperatura inicial
3. **Repetir**
4. **Repetir**
5. Gerar um vizinho S' aleatoriamente
6. $\Delta E = f(S') - f(S)$
7. **Se** $\Delta E \leq 0$ **então** $S = S'$, aceita a solução vizinha
8. **Senão** aceita com uma probabilidade $e^{\frac{-\Delta E}{T}}$ **Fim Se**
9. **Até** um determinado número de iterações à mesma temperatura T
10. $T = g(T)$, atualização da temperatura
11. **Até** critério de paragem satisfeito
12. **Devolver** melhor solução

Esta é geralmente calculada através da distribuição de Boltzmann (Blum e Roli, 2003), equação (7):

$$\mathrm{P} = \exp(-\frac{\mathrm{f(S')} - \mathrm{f(S)}}{\mathrm{T}}) \qquad (7)$$

onde, $f(S')$ e $f(S)$ representam o valor da função da solução objetivo de S' e S.

A escolha da temperatura é essencial para o desempenho do algoritmo. O fator de arrefecimento define o valor de T a cada iteração k. Este fator de redução de temperatura (α) é aplicado de modo a garantir uma redução constante da temperatura, tal como demostrado pela equação (8):

$$\mathrm{T_{k+1}} = \alpha \mathrm{T_k} \qquad (8)$$

Este processo é análogo ao processo de "cristalização" de metais e vidros, que quando arrefecido com a temperatura correta apresenta um gasto de energia menor, ou seja, a temperatura T diminui durante o processo de procura (Blum e Roli, 2003).

Aplicações desta Meta-heurística na resolução de problemas de escalonamento podem ser encontradas em, por exemplo (Yamada e Nakano, 1994), (Krishna et al., 1995), (Ponnambalam et al., 1999), (Steinhofel et al., 1999).

4.3 Metodologia de Seleção de Heurísticas

A Híper-heurística é um método de pesquisa heurística que procura automatizar, muitas vezes com a incorporação de técnicas de aprendizagem automática, o processo de seleção,

combinação ou adaptação de heurísticas mais simples para resolver de forma eficiente problemas de otimização. Uma das motivações para o estudo das Híper-heurísticas é o desenvolvimento de sistemas que consigam lidar com classes de problemas em vez de resolver apenas um problema (Burke et al., 2003b) (Ross, 2005).

No contexto deste trabalho adota-se a metodologia de seleção de heurísticas apresentadas em Burke et al. (2013) e já descrita anteriormente (Figura 5). Assim, apresentam-se algumas abordagens que procuram melhorar uma solução candidata através de um processo de seleção automática de heurísticas.

4.3.1 Abordagens para Seleção de Heurísticas

As técnicas de aprendizagem automática, *online* e *offline,* são de grande relevo para as estratégias de seleção de heurísticas, no sentido de permitir decisões de escolha informadas sobre qual a heurística a utilizar durante o processo de pesquisa. Metodologias Híper-heurísticas baseadas em heurísticas de perturbação foram aplicadas em problemas de otimização combinatória (Burke et al., 2003a) (Nareyek, 2003) (Ouelhadj e Petrovic, 2009, 2010).

Uma Híper-heurística seleciona uma heurística com base em dois fatores (Ozcan et al., 2008): (i) no método de seleção e (ii) no método de aceitação. Assim, uma das técnicas mais utilizadas como método de seleção é a Aprendizagem por Reforço.

Tabela 7 – Métodos de seleção e estratégias de aceitação de Heurísticas

Seleção de heurísticas com aprendizagem	
Aprendizagem por Reforço	(Nareyek, 2003) (Bai et al., 2007) (Pisinger e Ropke, 2007)
Aprendizagem por Reforço com Pesquisa Tabu	(Burke et al., 2003a) (Dowsland et al., 2007)
Função de Escolha	(Cowling et al., 2000) (Cowling et al., 2002)
Random Gradient	(Cowling et al., 2000) (Cowling et al., 2002)
Random Permutation Gradient	(Cowling et al., 2000) (Cowling et al., 2002)
Raciocínio Baseado em Casos	(Burke et al., 2002)
Estratégias de aceitação determinísticas	
Aceitar todas as soluções	(Cowling et al., 2000) (Cowling et al., 2002)
Aceitar apenas as melhores soluções	(Cowling et al., 2000) (Cowling et al., 2002)
Aceitar as soluções melhores ou iguais	(Cowling et al., 2000) (Cowling et al., 2002)

Um sistema baseado na Aprendizagem por Reforço interage com o ambiente, onde cada estado tem uma ação associada, que é realizada com base numa determinada política. O sistema procura aprender qual a ação a realizar com base na recompensa atribuía ao par estado-ação. No contexto da Híper-heurística, a recompensa resume-se a uma atribuição positiva ou negativa conforme o desempenho de heurística selecionada.

A estratégia de aceitação é um componente relevante para qualquer heurística de pesquisa local. Neste contexto, identificam-se duas estratégias (Burke et al., 2013): aceitação determinística ou não-determinística. Numa abordagem determinística é sempre tomada a mesma decisão de aceitação, independente do momento de decisão durante o processo de pesquisa. Por outro lado, numa abordagem não-determinística a decisão de aceitação pode ser diferente, dependendo do momento de decisão. Assim, apresenta-se na Tabela 7 alguns métodos de seleção bem como algumas estratégias de aceitação determinísticas, que serão utilizadas no âmbito deste trabalho.

Nareyek (2003) apresenta um método de seleção de heurísticas baseadas em Aprendizagem por Reforço. O autor, procura encontrar diferentes formas de selecionar heurísticas promissoras, de um conjunto de alternativas, durante o processo de pesquisa. A cada heurística é atribuído uma pontuação. A pontuação é alterado assim que a heurística é invocada e o desempenho avaliado. Se a heurística escolhida melhorar a função objetivo, a pontuação aumenta, caso contrário, a pontuação diminui. Após aplicada a heurística é utilizada uma estratégia de aceitar todas a soluções. Outros exemplos, podem ser encontrados em (Bai et al., 2007) e (Pisinger e Ropke, 2007).

Burke et al. (2003a) apresenta um método de seleção baseado em Aprendizagem por Reforço com Pesquisa Tabu. De forma similar ao estudo realizado por Nareyek (2003), as heurísticas de baixo nível são selecionadas com base numa pontuação. A Híper-heurística proposta pelos autores incorpora uma lista tabu de heurísticas de baixo nível, que temporariamente se encontram excluídas das heurísticas disponíveis. O algoritmo seleciona, de forma determinística, a heurística com maior pontuação e que não se encontra na lista tabu. A heurística selecionada é utilizada independentemente de melhorar ou não a função objetivo (estratégia de aceitar todas a soluções). Se a heurística melhorar a função objetivo, a pontuação aumenta, caso contrario, a pontuação diminui e a heurística é colocada na lista tabu. Burke et al. (2005) estende esta metodologia com uma lista tabu de tamanho fixo para a resolução de problemas de otimização multiobjetivo de alocação de espaço e tabelas de horários.

Cowling et al. (2000) descreve uma abordagem baseada na classificação estatística de heurísticas de baixo nível. Neste método, a informação histórica sobre o desempenho das heurísticas de baixo nível é acumulado numa função de escolha. A escolha de uma heurística de baixo nível depende do valor da função de escolha. Os autores definem a função de escolha como, "*a chave para capturar a natureza da região do espaço de soluções que se encontra sobre exploração e decidir qual a vizinhança (heurística de baixo nível) a escolher a seguir, com base no desempenho histórico de cada vizinhança*". A função de escolha representa a soma ponderada do desempenho de cada heurística de baixo nível, da eficácia dos pares consecutivos de heurísticas de baixo nível, e da quantidade de tempo desde a última heurística a ser

escolhida. Como estratégia de aceitação, foram considerada duas abordagens: aceitar todas as soluções e aceitar apenas as melhores. Os resultados experimentais em Cowling et al. (2000) mostram que a Híper-heurística com a estratégia de aceitação de todas as soluções é promissor.

Em (Cowling et al., 2000, 2002), os autores propuseram e compararam uma variedade de Híper-heurísticas aplicadas a dois problemas de escalonamento. As Híper-heurísticas utilizam diferentes métodos de seleção. Um método simples de seleção aleatória de heurísticas de baixo nível. O *Random Gradient*, que é uma variante do método de seleção aleatório, seleciona uma heurística aleatoriamente e aplica-a repetidamente até à estagnação do processo de pesquisa, ou seja não apresenta soluções melhores. Por outro lado, o *Random Permutation* gera uma ordenação aleatória das heurísticas de baixo nível e a cada nova iteração aplica uma heurística, respeitando a ordem definida pelo processo de ordenação. O *Random Permutation Gradient*, que é uma variante do *Random Permutation*, funciona da mesma forma até ao momento em que a solução não apresenta melhorias, mudando a ordem de ordenação as heurísticas. Embora os métodos de seleção, *Random Gradient* e *Random Permutation Gradient,* utilizem mecanismos de aleatoriedade, podem ser considerados métodos de seleção de heurísticas inteligentes, pois incorporam Aprendizagem por Reforço. Ou seja, estes métodos incorporam o conceito de recompensa, atribuindo uma recompensa em conformidade com o desempenho da heurística.

Burke et al. (2002) desenvolveu uma Híper-heurística que utiliza Raciocínio Baseado em Casos, para seleção de heurísticas de baixo nível, aplicado ao problema de planeamento do calendário de exames. A abordagem tem semelhanças com aquela apresentada em (Ross et al., 2002), que utiliza um sistema de classificação de aprendizagem. O calendário de exames é construído passo a passo, através da aplicação, em cada iteração, de uma heurística de baixo nível na solução parcial. A heurística mais adequada é escolhida a partir de uma base de casos. A base de casos é uma coleção de casos, onde cada caso descreve uma possível solução parcial e sugere uma heurística de baixo nível que foi previamente encontrado, e referenciada como sendo eficaz para lidar com a solução parcial em análise. A solução parcial em cada caso é representada por uma lista de características, que é determinado durante a fase de descoberta de conhecimento, por um procedimento específico de Pesquisa Tabu. Em cada fase do processo de construção do calendário, a Híper-heurística identifica o caso que apresenta o maior grau de semelhança com solução parcial em análise, e aplica a heurística de baixo nível que corresponde ao caso escolhido (Chakhlevitch e Cowling, 2008).

4.4 Áreas de Aplicação

A pesquisa heurística é estudado em áreas como, as ciências da computação e Inteligência Artificial. O desenvolvimento de melhores técnicas de pesquisa passa pela integração de componentes de aprendizagem que podem de forma adaptativa direcionar a pesquisa. Muitas técnicas têm emergido de forma independente que exploram alguma forma de aprendizagem, ou pesquisa num espaço de soluções, para melhorar a resolução de problemas e tomada de

decisão. Assim, descreve-se de seguida algumas dessas abordagens, classificando entre abordagens *online* e *offline* (Burke et al., 2013).

4.4.1 Abordagens Offline

Configuração de Algoritmos: definição da parametrização de um algoritmo conforme o problema em análise. É um problema comum em problemas de otimização, onde a função objetivo captura o desempenho dum algoritmo sobre um conjunto de instâncias (Hutter et al., 2007). Dependendo do número e tipo de parâmetros, os métodos utilizados para resolver este problema de otimização incluem enumeração exaustiva, pesquisa local, pesquisa local iterada, entre outros (Burke et al., 2013).

Meta-Aprendizagem: a meta-aprendizagem foi inspirada no problema de seleção algorítmica formulada por Rice (1976). Reconhecendo este problema como uma tarefa de aprendizagem, a comunidade de aprendizagem automática desenvolveu abordagens de meta-aprendizagem, principalmente para aprender sobre problemas de classificação. A generalização dos conceitos da meta-aprendizagem à restrição, satisfação e otimização está intimamente relacionada com a investigação das Híper-heurística (Burke et al., 2013). Exemplos de investigação podem ser encontrados em (Smith-Miles, 2008).

4.4.2 Abordagens Online

Controlo de parâmetros em algoritmos evolucionários: abordagem para configuração *online* de algoritmos, onde o objetivo é parametrizar o algoritmo em tempo de execução (Lobo et al., 2007).

Seleção de Operadores Adaptativos: o objetivo é encontrar estratégias *online*, de forma a permitir a seleção automática de diferentes operadores. A seleção de operadores de forma adaptativa é realizado através de dois mecanismos: atribuição da recompensa, que define a recompensa atribuída ao operador e seleção do operador, que define o operador a ser selecionado com base no desempenho anterior (Maturana et al., 2009).

Pesquisa Reativa: é uma metodologia *online* que defende a integração de técnicas sub-simbólicas de aprendizagem automática na pesquisa de heurísticas para resolver problemas complexos de otimização (Battiti e Brunato, 2007). A técnica de aprendizagem automática atua como uma camada de nível superior para permitir a auto-parametrização do algoritmo durante o processo de pesquisa.

Pesquisa da Vizinhança Variável (PVV): Embora geralmente não incluam mecanismos de adaptação, a PVV (Mladenovic e Hansen, 1997), está relacionada com a seleção heurística baseada em heurísticas perturbativas que possam explorar o poder da pesquisa em múltiplas vizinhanças. A PVV muda sistematicamente entre vizinhanças seguindo uma sequência pré-definida de modo a que a pesquisa possa alargar a exploração de vizinhanças da solução atual.

4.5 Sumário do Capítulo

Neste capítulo foi apresentada uma visão geral de uma metodologia de pesquisa e otimização, denominada de Híper-heurística. Um dos principais objetivos desta abordagem é elevar o nível de generalidade em que os sistemas de otimização podem operar. A Híper-heurística é uma abordagem de alto nível, onde dado um conjunto de Meta-heurísticas, procura selecionar a mais adequada para solucionar um determinado problema.

Neste contexto, foi feita uma descrição mais detalhada sobre as Meta-heurísticas utilizadas, nomeadamente os Algoritmos Genéticos, a Colónia Artificial de Abelhas, a Otimização por Colónia de Formigas, o *Particle Swarm Optimization*, a Pesquisa Tabu e o *Simulated Annealing*.

Para além disso, também foi apresentada a metodologia de seleção utilizada pela Híper-heurística para selecionar uma Meta-heurística, bem como a estratégia de aceitação das soluções alcançadas pela aplicação de uma Meta-heurística a um problema.

Capítulo 5

Sistema AutoDynAgents e o Módulo da Híper-Heurística

O sistema AutoDynAgents (*Autonomic Agents with Self-Managing Capabilities for Dynamic Scheduling Support in a Cooperative Manufacturing System*) (Madureira e Pereira, 2010) (Madureira et al., 2011a) consiste num Sistema Multiagente (SMA) para a resolução de problemas de Escalonamento sujeitos a perturbações. Este sistema utiliza técnicas de otimização para a obtenção de soluções quase-ótimas dos planos de escalonamento, nomeadamente Meta-heurísticas, tais como Algoritmos Genéticos, Colónia Artificial de Abelhas, Otimização por Colónia de Formigas, *Particle Swarm Optimization*, Pesquisa Tabu e *Simulated Annealing*. A aplicação destas técnicas requer um conhecimento prévio da parametrização, adequadas ao problema a tratar. Esta especificação dos parâmetros a usar é uma tarefa bastante difícil e requer alguma experiência e conhecimento pericial. Neste capítulo, será proposto um mecanismo de aprendizagem com base numa Híper-heurística para a parametrização automática das Meta-heurísticas a usar na resolução de problemas de escalonamento.

Assim, será descrito o sistema AutoDynAgents e a sua arquitetura de agentes. Será também descrita a estrutura do AutoDynAgents com a Híper-heurística, onde se inclui a Interface Gráfica e o Módulo de Escalonamento. Por fim, será descrita a Híper-heurística desenvolvida e demonstrado um exemplo ilustrativo, para uma melhor perceção do funcionamento do sistema.

5.1 Sistemas Multiagente

Os SMA são compostos pela interação de múltiplos elementos computacionais, conhecidos como agentes. Os agentes são sistemas computacionais dotados de duas capacidades fundamentais. Primeiro, pelo menos até um certo ponto, são capazes de efetuar ações autónomas, ou seja, decidirem autonomamente o que fazer em ordem a satisfazer os objetivos para os quais foram criados. Segundo, são capazes de interagir com outros agentes, não só pela simples troca de dados, mas também através de algumas capacidades sociais: cooperação, coordenação e negociação. Em (Panait e Luke, 2005) um agente é definido como "*um mecanismo computacional que exibe um elevado grau de autonomia, que executa ações no seu ambiente baseado em informação (sensores, feedback) recebida a partir do ambiente*".

Embora o agente faça parte do ambiente, é considerado como possuidor de características adicionais ao ambiente. Isto deve-se ao facto de um agente ser uma entidade independente

com os seus próprios objetivos, ações e conhecimento. Num ambiente com um único agente, nenhuma outra entidade é reconhecida pelo agente. Como tal, se realmente existirem outros agentes no ambiente, estes são simplesmente modelados como não tendo objetivos e considerados como parte do ambiente. Contrariamente, nos SMA existem vários agentes a modelar objetivos e ações de outros agentes. Num típico cenário multiagente, existem interações entre os agentes, isto é, comunicação entre si, tal como demonstrado na Figura 7.

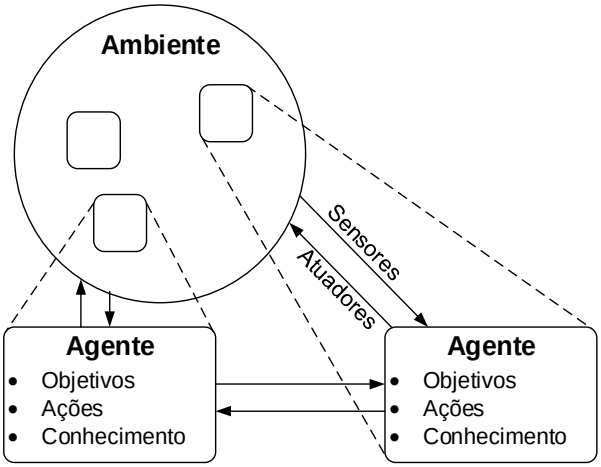

Figura 7 – Cenário Multiagente (adaptado de (Stone e Veloso, 2000))

No cenário da Figura 7, os agentes modelam os objetivos, as ações, e o conhecimento uns dos outros, podendo estes serem diferentes. Os agentes também podem interagir (comunicar) diretamente, como indicado pelas setas que se encontram entre os agentes.

Um ambiente multiagente é definido em (Panait e Luke, 2005) como aquele no qual *"existe mais do que um agente, que interagem uns com os outros, onde existem restrições no ambiente tal que os agentes não possam, num dado momento, saber tudo que os outros agentes sabem sobre o mundo (incluindo o estado interno dos outros agentes)".* Estas restrições são, segundo Panait e Luke (2005), importantes para a noção da definição do problema de um Sistema Multiagente. Caso contrário, os agentes distribuídos podem atuar em sincronização, sabendo exatamente a situação em que estão os outros agentes e quais os comportamentos que irão efetuar. Esta "omnisciência" permite que os agentes atuem como se fossem meros apêndices pertencentes a um único controlador. Adicionalmente, se o domínio não exigir qualquer interação, então este poderá ser decomposto em tarefas separadas e totalmente independentes, com cada agente único a resolver uma tarefa.

Os SMA podem ser utilizados em problemas cuja complexidade torna o processo de resolução intratável para os seres humanos. Assim, o recurso a técnicas de decomposição de problemas permitem distribuir a computação, e os problemas resultantes, em formas mais simples, mapeadas num conjunto de agentes distintos que interagem com a finalidade de alcançar soluções satisfatórias.

5.2 Sistema AutoDynAgents

O sistema AutoDynAgents é um sistema que permite a resolução de problemas de escalonamento dinâmicos com capacidades autónomas, em que um conjunto de agentes modela um sistema real de fabrico sujeito a perturbações. O sistema é capaz de encontrar soluções ótimas ou próximas do ótimo através da utilização de Meta-heurísticas, lidar com o dinamismo (chegada de novas tarefas, tarefas canceladas, alteração dos atributos das tarefas, etc.), alterar/adaptar os parâmetros do algoritmo de acordo com a atual situação, alternar entre Meta-heurísticas, e realizar a coordenação entre os agentes, através da cooperação e negociação (Madureira et al., 2014) (Madureira et al., 2013).

A abordagem utilizada pelo AutoDynAgents para a resolução de problemas de escalonamento é diferente das encontradas na literatura, uma vez que, neste sistema, existe um grupo de agentes de recursos, cada um responsável por otimizar o plano da máquina correspondente através da aplicação das Meta-heurísticas, obtendo soluções locais. O agente coordenador é responsável por reunir e integrar as soluções locais e construir um único plano de escalonamento, através da aplicação de um mecanismo de coordenação (Madureira et al., 2008) (Madureira et al., 2011b).

A arquitetura do sistema AutoDynAgents assenta em seis diferentes tipos de agentes. Esta é uma arquitetura híbrida, uma vez que os agentes podem definir os seus planos de escalonamento, reagir a mudanças e cooperar entre eles. Neste contexto, apresenta-se de seguida a descrição dos agentes (Madureira e Pereira, 2011) (Madureira et al., 2011b):

- **Agente UI**: é responsável pela interface e também pela geração dinâmica dos agentes de tarefas e de recursos necessários, de acordo com o número de *jobs* e máquinas que compõem o problema de escalonamento, e atribui a cada tarefa o respetivo agente tarefa. Para além disso, também é responsável pela coordenação das soluções enviadas pelos Agentes Recurso, utilizando um mecanismo de coordenação no caso de as soluções serem exequíveis;
- **Agentes Tarefas**: são responsáveis pela geração dos tempos de processamento das tarefas e atribuição automática de cada operação da tarefa ao respetivo Agente Recurso;
- **Agentes Recurso**: são responsáveis pelo escalonamento das operações que requerem processamento na máquina supervisionado pelo agente. Estes agentes implementam as Meta-heurísticas, de forma a encontrar o melhor plano de operações. No fim do processo comunicam o resultado ao Agente UI;
- **Agente Auto-Configuração**: é responsável pela monitorização do sistema, com o objetivo de detetar mudanças ocorridas no escalonamento, permitindo que o sistema execute uma adaptação dinâmica;
- **Agente Auto-Otimização**: é responsável pela parametrização automática das Meta-heurísticas, de acordo com as características dos problemas e comportamento do

sistema. O agente recebe o problema inicial e automaticamente escolhe a Meta-heurística e a parametrização a utilizar;

- **Agente Auto-Reparação**: dota o sistema com a capacidade de diagnosticar desvios das condições normais e de forma pró-ativa desenvolve ações para normalizá-los e evitar interrupções de serviço.

Os agentes Auto-* (Auto-Configuração, Auto-Otimização, e Auto-Recuperação) existem para que o conceito de Computação Autónoma (*Autonomic Computing*) seja aplicável ao sistema (Kephart e Chess, 2003) (Parashar e Hariri, 2006). Estas capacidades de auto-gestão são globais ao sistema, sendo que não existe um módulo único para cada agente. Assim, cada agente do sistema tem o seu módulo de Auto-Configuração, Auto-Otimização e Auto-Recuperação através de comunicações/monitorizações do respetivo agente de auto-gestão (Pereira, 2009, 2014).

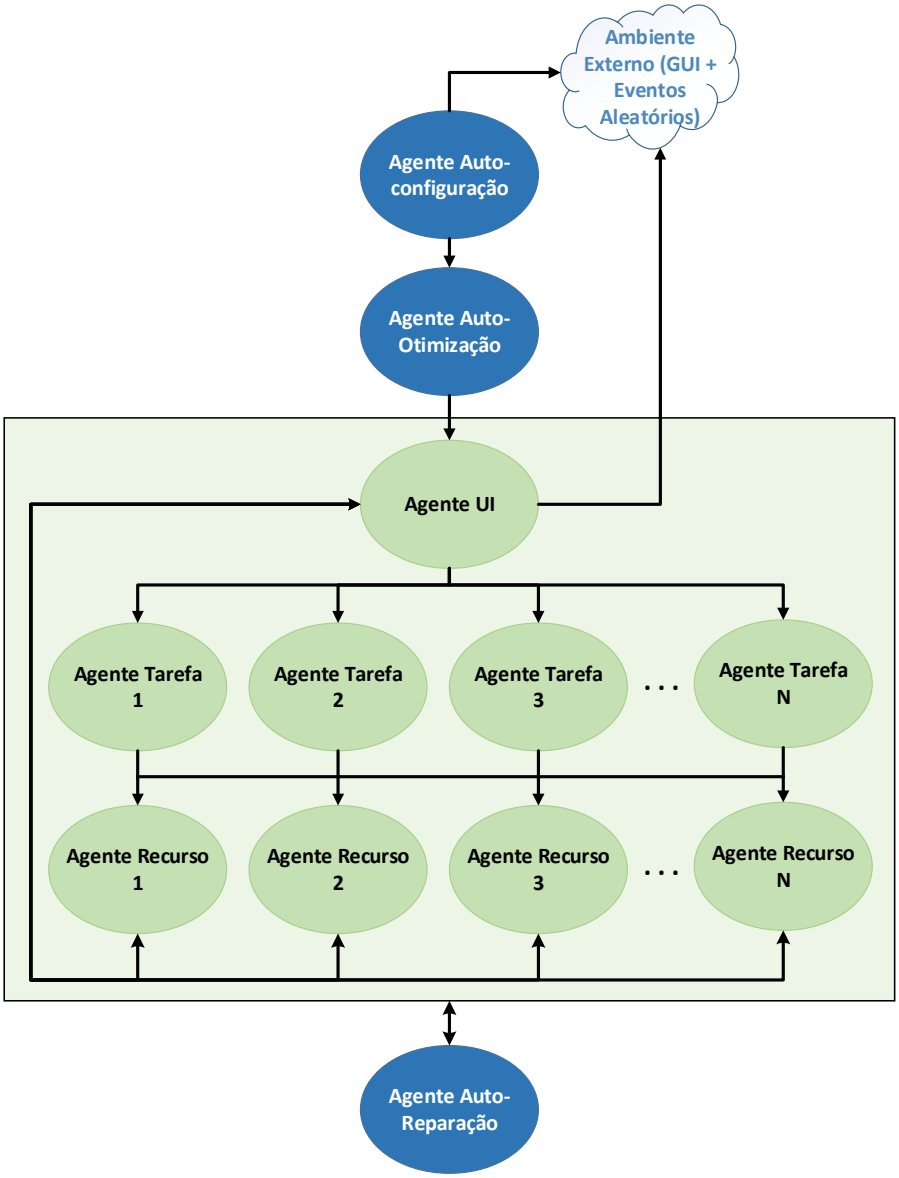

Figura 8 – Arquitetura dos agentes do sistema AutoDynAgents (Pereira e Madureira (2013))

Para uma melhor compressão apresenta-se na Figura 8 a arquitetura dos agentes dos sistema AutoDynAgents, onde é possível confirmar a interligação existente entre os vários agentes presentes na arquitetura, com destaque para a comunicação entre os agentes do Sistema Multiagente.

O sistema Multiagente é constituído por um Módulo de Escalonamento, responsável por gerar o plano de escalonamento apresentado na Interface Gráfica. Neste contexto, descreve-se de seguida a estrutura do Sistema AutoDynAgents com a Híper-heurística, implementada no agente Auto-Otimização.

5.2.1 Estrutura do Sistema AutoDynAgents com a Híper-heurística

A partir da Figura 9 é possível verificar a ligação existente entre os diferentes módulos. O módulo de Interface Gráfica (GUI) é responsável pela definição do problema e parametrização inicial das Meta-heurísticas. O módulo da Híper-heurística é o responsável por selecionar, em tempo de execução, a Meta-heurística a aplicar a um determinado problema com a respetiva parametrização. Para além disso, o sistema também é constituído pelo módulo de Escalonamento, que como referido anteriormente, é responsável por gerar o plano de escalonamento que é apresentado na Interface Gráfica.

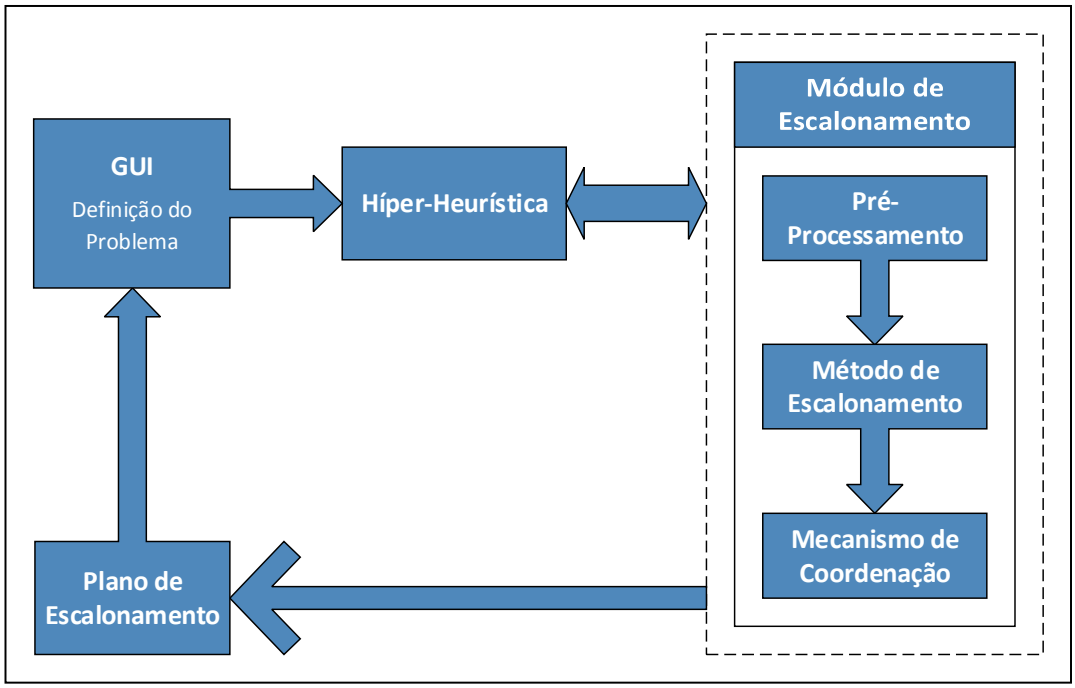

Figura 9 – Estrutura do sistema AutoDynAgents com a Híper-heurística

Assim, de seguida serão descritos com mais detalhe cada um dos módulos apresentados na estrutura do sistema (Figura 9).

5.2.1.1 Interface Gráfica

A Interface Gráfica, como referido, é responsável pela definição dos dados do problema, pela parametrização inicial das Meta-heurísticas, e pela visualização dos resultados obtidos. Assim, a interface disponibiliza as seguintes funcionalidades (Madureira et al., 2008):

- Definição/carregamento dos dados do problema;
- Visualização do grafo de operações de cada tarefa;
- Definição da parametrização base das Meta-heurísticas para o caso de não existirem registos anteriores similares;
- Comunicação com o Sistema Multiagente (envio dos dados e obtenção da solução);
- Visualização dos resultados obtidos: gráfico de *Gantt* do plano de escalonamento obtido, gráfico comparativo entre o plano previsto e o plano elaborado, gráfico comparativo entre os pesos das tarefas, gráfico comparativo entre os atrasos das tarefas e ainda um relatório de escalonamento.

O processo inicia-se pelo carregamento dos dados de entrada. De seguida, define-se a parametrização pretendida para as Meta-heurísticas. Por fim, é iniciada a comunicação com o sistema de forma a resolver o problema. Quando o processo termina é possível visualizar os resultados obtidos. Como tal, é possível analisar gráficos comparativos (Figura 10), gráfico de *Gantt* do plano de escalonamento (Figura 11), relatório de escalonamento (Figura 12) e o relatório de resultados da execução da Meta-heurística (Figura 13).

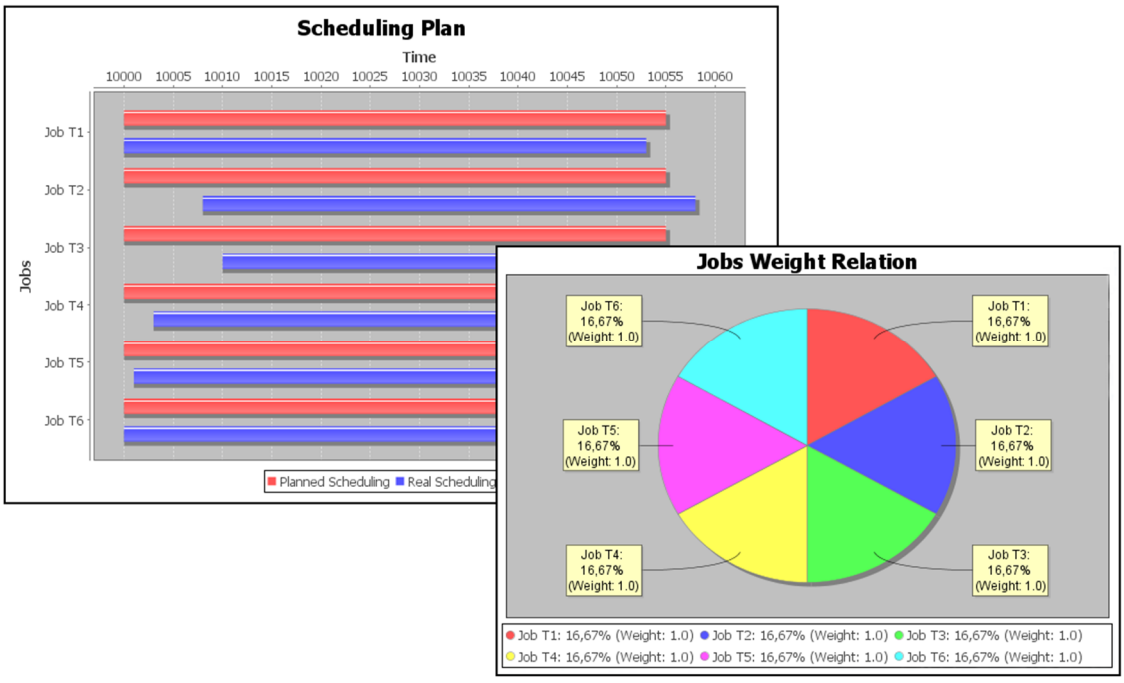

Figura 10 – Gráfico comparativo entre o plano previsto e o gerado, e gráfico comparativo dos pesos das tarefas

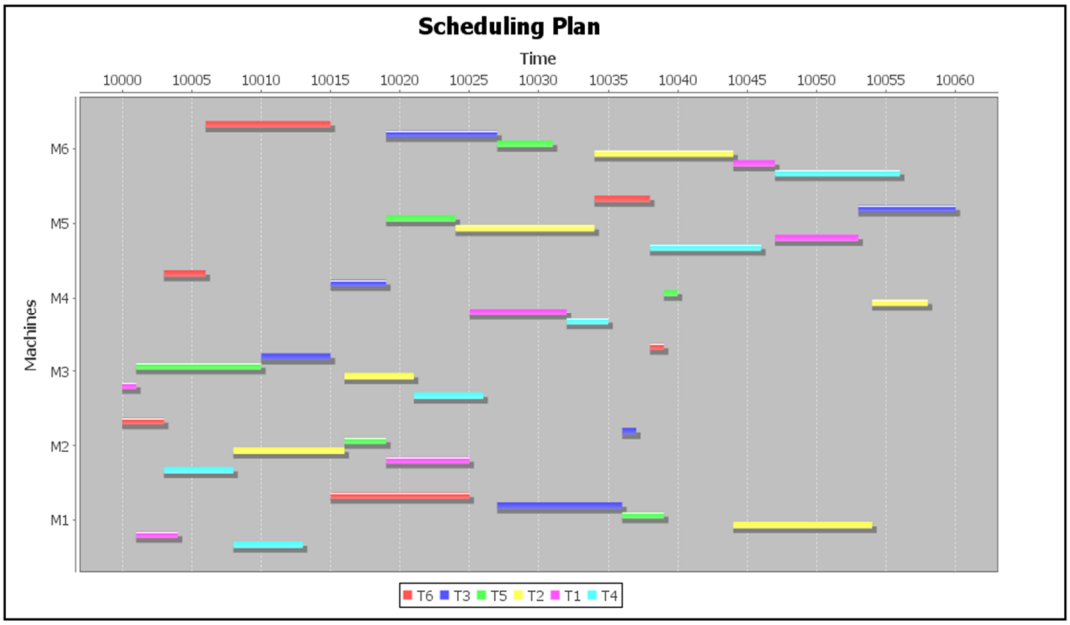

Figura 11 – Gráfico de Gantt do escalonamento

Job	Weight	Rel. Date	Due Date	Start	End	Fj	T	WT
T1	1	0	55	0.0	53.0	53.0	0.0	0.0
T2	1	0	55	8.0	58.0	50.0	3.0	3.0
T3	1	0	55	10.0	60.0	50.0	5.0	5.0
T4	1	0	55	3.0	56.0	53.0	1.0	1.0
T5	1	0	55	1.0	40.0	39.0	0.0	0.0
T6	1	0	55	0.0	39.0	39.0	0.0	0.0

Summary

CMax	TMax	N	Sum_Fj	Sum_Tj	Sum_wjCj	Sum_wjTj	Fmax	U	Time (s)
60.0	5.0	3.0	197.0	9.0	306.0	9.0	53.0	54.73	1.19

Figura 12 – Relatório de escalonamento

Results

Run num.	CMax	TMax	N	Sum_Fj	Sum_Tj	Sum_wjCj	Sum_wjTj	Fmax	U	Time (s)
1	60.0	5.0	3.0	197.0	9.0	306.0	9.0	53.0	54.73	1.19
2	60.0	5.0	2.0	197.0	6.0	327.0	6.0	56.0	54.73	0.89
3	64.0	9.0	5.0	197.0	25.0	341.0	25.0	64.0	51.31	0.55
4	66.0	11.0	4.0	197.0	31.0	352.0	31.0	62.0	49.75	5.23
5	60.0	5.0	2.0	197.0	6.0	318.0	6.0	59.0	54.73	0.74

Best, Worst & Average Results

Run num.	CMax	TMax	N	Sum_Fj	Sum_Tj	Sum_wjCj	Sum_wjTj	Fmax	U	Time (s)
Best	60.0	5.0	2.0	197.0	6.0	327.0	6.0	56.0	54.73	0.55
Worst	66.0	11.0	4.0	197.0	31.0	352.0	31.0	62.0	49.75	5.23
Average	62.0	7.0	3.21	197.0	15.41	328.81	15.41	58.8	53.05	1.72

Figura 13 – Resultado da execução da Meta-heurística

Na Figura 13, também é possível analisar medidas estatísticas dos resultados obtidos, bem como os respetivos tempos de execução. Para além disto, também é possível inserir eventos após a receção dos dados, por exemplo, inserir novas tarefas, cancelar tarefas e alterar atributos das tarefas.

5.2.1.2 Módulo da Híper-heurística

A Híper-heurística desenvolvida é responsável pela parametrização automática dos parâmetros das Meta-heurísticas, conforme o problema em análise. O método recebe o problema inicial, seleciona automaticamente a Meta-heurística a utilizar, define a parametrização, e comunica com o sistema Multiagente. A auto-parametrização das Meta-heurísticas é realizada através de aprendizagem por experiência, uma vez que utiliza o *Q-Learning*, uma técnica de Aprendizagem por Reforço. Assim, sempre que um novo problema surge no sistema, a Híper-heurística utiliza a experiência adquirida para definir a Meta-heurística e respetiva parametrização a utilizar. Quando um novo problema é resolvido, e apresenta um melhor resultado que o anterior, é armazenado para uso futuro. Este módulo é descrito com maior detalhe na subsecção 5.3.

5.2.1.3 Módulo de Escalonamento

O módulo de escalonamento é constituído pelo Método de Escalonamento baseado em Meta-heurísticas, proposto por Madureira (2003). As soluções geradas são representadas diretamente, onde o plano de escalonamento é descrito como uma sequência de operações, isto é, cada posição representa um índice de operação com tempos de processamento iniciais e finais. Cada operação é caracterizada pelo índice (i, j, k, l), onde i define a máquina onde a operação k é processada, a tarefa j a que pertence, e o nível l no grafo de precedências de operações (nível 1 corresponde às operações iniciais, sem precedentes) (Madureira, 2003). O Método de Escalonamento é implementado em duas fases, como descrito na Tabela 8.

O processo inicia-se pela decomposição em problemas de máquina única[6], onde são conhecidos os tempos de lançamento e datas de entrega das tarefas. Assim, com esta informação é possível determinar as datas de entrega e lançamento para cada problema de máquina única, sendo que cada problema é resolvido pela aplicação de uma Meta-heurística. No fim do processo, as soluções individuais são integradas de modo a se obter uma solução global para o problema inicial.

No entanto, a integração de várias máquinas única pode originar planos inviáveis para o problema de escalonamento. Isto acontece, pois a inter-relação das máquinas não é tida em conta durante processo, como tal é necessário realizar a coordenação das soluções antes da integração num único plano. Esta função fica a cargo do Mecanismo de Coordenação, que tem como objetivo reparar a solução obtida, através da coordenação dos tempos de ocupação das máquinas e das relações de precedência entre operações, redefinindo ou confirmando os

[6] É o processo de atribuição a um conjunto de tarefas uma máquina única ou recurso. As tarefas são organizadas de modo a que uma ou várias medidas de desempenho possam ser otimizadas.

tempos de início e de conclusão para cada operação de modo a ser obtido um plano de escalonamento exequível (Madureira, 2003) (Madureira et al., 2008).

Tabela 8 – Método de Escalonamento baseado em Meta-heurísticas (Madureira, 2003)

Primeira Fase	**Encontrar um plano para o problema, baseado na sua decomposição nos vários problemas de máquina única que o constituem.**
Passo 1	Determinar os tempos de conclusão estimados C_{ijkl}, em que as operações deverão ser concluídas de modo que as datas de entrega das tarefas sejam cumpridas.
Passo 2	Determinar o intervalo dos tempos de início estimados $[t_{ijkl}, T_{ijkl}]$, em que as operações deverão ser iniciadas de modo que as datas de entrega das respetivas tarefas sejam cumpridas.
Passo 3	Definir todos os problemas de máquina única baseados na informação calculada no passo 1 e 2.
Passo 4	Aplicar uma Meta-heurística a cada um dos problemas de máquina única.
Passo 5	Integrar as soluções obtidas no problema principal.
Segunda Fase	**Verificar a exequibilidade do plano obtido, e aplicar o mecanismo de coordenação se necessário.**
Passo 6	Verificar se estamos perante uma solução válida ou exequível. Se não for válida então é necessário aplicar o Mecanismo de Coordenação.

5.3 Módulo da Híper-Heurística

As heurísticas são técnicas simples e rápidas, capazes de encontrar soluções suficientemente boas para resolver problemas de otimização combinatória. São frequentemente utilizadas quando os métodos exatos se revelam inadequadas, muitas vezes por questões temporais. As Meta-heurísticas e as Híper-heurísticas são métodos de pesquisa mais genéricos e inteligentes.

As Meta-heurísticas, descritas na secção 4.2, são métodos utilizados para resolver problemas de otimização combinatória. Utilizam uma função objetivo para guiar a pesquisa de forma inteligente na procura por melhores soluções. Em comparação com as heurísticas, as Meta-heurísticas utilizam estratégias de nível superior para evitar os ótimos locais. O objetivo, destes métodos é atingir o ótimo global, ou seja, a melhor solução entre todos os ótimos locais. No entanto, para que estas soluções ótimas, ou quase ótimas possam ser atingidas, é necessário que estes algoritmos sejam parametrizados da forma correta, tarefa difícil, que muitas vezes passa pela utilização do método tentativa-erro.

As Híper-heurísticas, descritas no capítulo 4, são métodos de pesquisa que operam a um nível superior. Ao contrário das Meta-heurísticas, as Híper-heurísticas realizam a pesquisa num espaço de (meta) heurísticas. Normalmente, as Híper-heurísticas não utilizam qualquer tipo de

conhecimento específico do domínio do problema. Apenas informação sobre o desempenho da (meta) heurística de baixo nível, por exemplo, o melhor tempo de execução ou obtenção do ótimo global. Desta forma as Híper-heurísticas tendem a ser mais genéricas do que as meta-heurísticas, usando a informação que é comum a todos os problemas de otimização.

O sistema AutoDynAgents tem como objetivo dotar o sistema com a capacidade de auto parametrização das Meta-heurísticas, de acordo com o problema a ser resolvido. O sistema deve ser capaz de escolher uma Meta-heurística a utilizar e definir os parâmetros, de acordo com o problema atual. Para além disso, pode ser possível permutar de um método para o outro, dependendo do problema e da experiência acumulada. Esta otimização é realizada através da aprendizagem e experiência.

Existe uma relação direta entre a Aprendizagem por Reforço e os algoritmos de pesquisa. Dado um espaço de estados, um espaço de ações e uma função de recompensa, o problema de Aprendizagem por Reforço, pode ser reduzida a uma pesquisa num espaço de políticas. Este pode ser visto como um problema de otimização estocástica *online* com uma função objetivo desconhecida. Assim, questões semelhantes, como a exploração e o aproveitamento, são muitas vezes chamadas de diversificação e intensificação no domínio das Meta-heurística.

Os métodos de Aprendizagem por Reforço podem ser utilizados a diferentes níveis para resolver problemas de otimização combinatória. Podem ser aplicados diretamente ao problema, como componente de uma Meta-heurística ou como componente de uma Híper-heurística. Assim, no âmbito deste trabalho foi desenvolvido uma Híper-heurística que incorpora um método de Aprendizagem por Reforço (subcapítulo 3.2.3), o algoritmo *Q-Learning*, descrito na secção 3.3.3. A Híper-heurística é capaz de configurar os parâmetros das Meta-heurísticas, conforme os diferentes problemas vão surgindo no sistema.

Neste sentido, sempre que surge um novo problema para tratar é recuperada uma lista de registos, armazenados numa base de dados, que são utilizados como espaço de pesquisa pela Híper-heurística. Os registos incorporam conhecimento específico sobre problemas já resolvidos, sendo utilizado para resolver um novo problema. A cada registo está associada uma Meta-heurística, bem como a parametrização que originou a resolução. A Híper-heurística seleciona um registo e aplica a Meta-heurística com a respetiva parametrização ao problema em análise. A solução alcançada, se for melhor que o anterior, é armazenada na base dados para futura utilização.

Seguidamente, serão descritos os vários aspetos da implementação da Híper-heurística, nomeadamente a base de dados que armazena os problemas já resolvidos e o algoritmo *Q-Learning*.

5.3.1 Base de Dados

A Base de Dados utilizada para armazenar os resultados obtidos pela execução das Meta-heurísticas escolhidas pelo algoritmo *Q-Learning* é composta por sete tabelas e encontra-se

ilustrada na Figura 14. Esta base de dados é utilizada como suporte no processo de aprendizagem. Ou seja, armazena problemas já resolvidos que são utilizados na resolução de novos problemas.

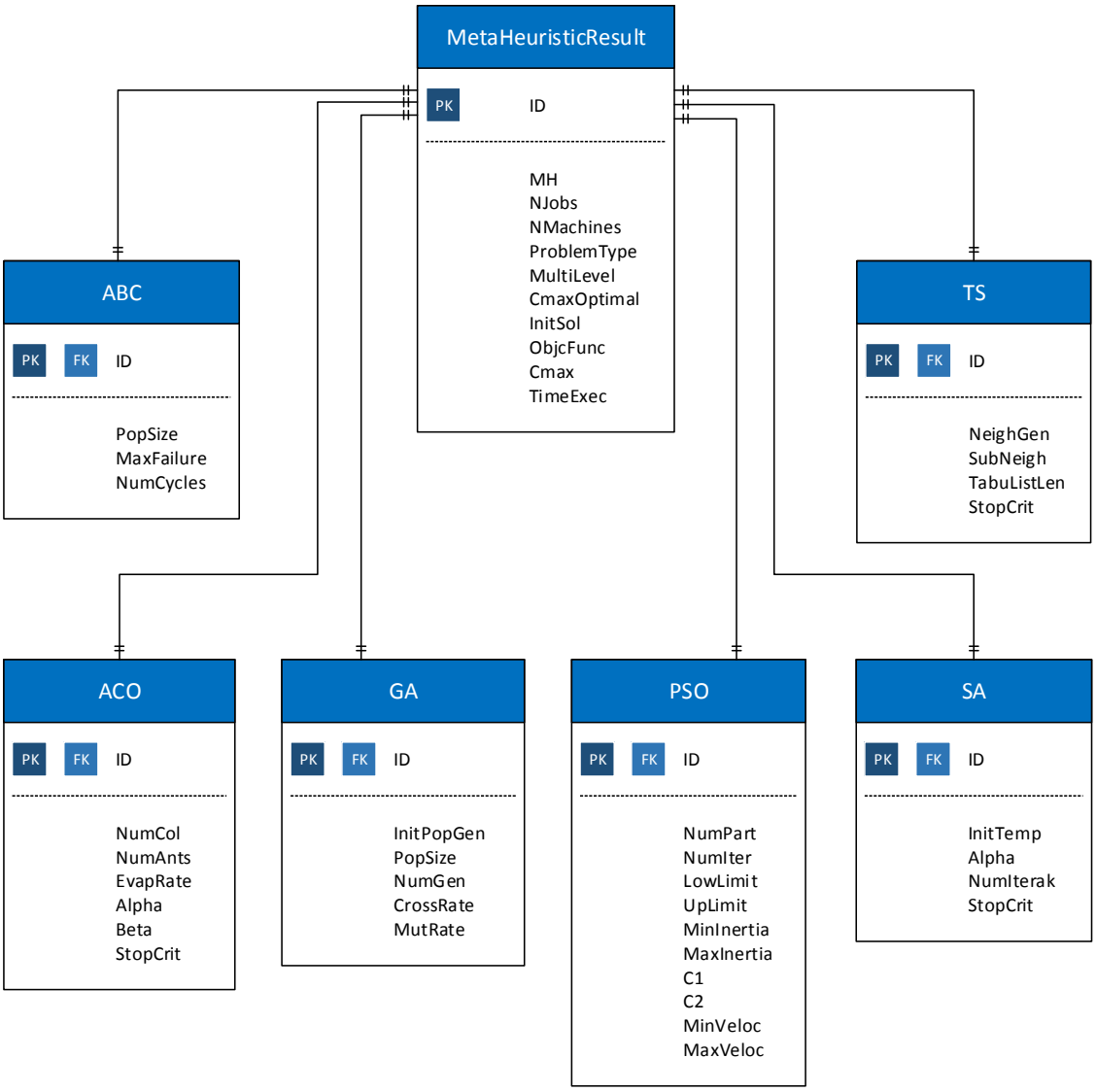

Figura 14 – Base de dados para armazenar os resultados obtidos

Na tabela *MetaHeuristicResult* são armazenados os resultados gerados pelo sistema. Esta tabela guarda os tempos de conclusão e de execução obtidos. Para além disso, também armazena a solução inicial e a função objetivo sugerida pelo algoritmo *Q-Learning*, atributos estes comuns a todas as tabelas. Na Tabela 9 encontram-se descritos os atributos e os tipos de dados utilizados.

Tabela 9 – Campos da tabela MetaHeuristicResult

MetaHeuristicResult		
Atributo	**Tipo de Dados**	**Descrição**
ID	Integer	Identificador único, que incrementa automaticamente
MH	String	Meta-heurística a utilizar (ABC, ACO, GA, PSO, SA ou TS)
NJobs	Integer	Número de tarefas do problema
NMachines	Integer	Número de máquinas do problema
ProblemType	String	Tipo de problema (*Single-Machine*, *Open-Shop*, *Flow-Shop*, ou *Job-Shop*)
MultiLevel	Boolean	Indica se o problema tem operações com mais de que um precedente
CmaxOptimal	Integer	Valor ótimo do tempo de conclusão (C_{max}) conhecido
InitSol	String	Regra de prioridade utilizada para construir a solução inicial (SeqNível, EDD, SPT, REDD, ou RND)
ObjFunc	String	Função objetivo utilizada para a minimização do problema (C_{max} ou WT).
Cmax	Integer	Tempo de conclusão (C_{max}) obtido para um determinado problema
TimeExec	Double	Tempo computacional de execução de um determinado problema

As tabelas seguintes armazenam a parametrização/solução para cada resultado gerado pela aplicação de uma Meta-heurística a um determinado problema. Assim, para cada Meta-heurística utilizada no sistema existe uma tabela, isto é, *Artificial Bee Colony* (ABC), Otimização por Colónia de Formigas (ACO), Algoritmos Genéticos (GA), *Particle Swarm Optimization* (PSO), *Simulated Annealing* (SA), e Pesquisa Tabu (TS), sendo que os atributos encontram-se descritos na Tabela 10, Tabela 11, Tabela 12, Tabela 13, Tabela 14 e Tabela 15.

Tabela 10 – Campos da tabela ABC

ABC		
Atributo	**Tipo de Dados**	**Descrição**
PopSize	Integer	Define o tamanho da população
MaxFailure	Integer	Define a distância que a abelha pode percorrer, antes de ter que voltar à colónia
NumCycles	Integer	Número de iterações

Tabela 11 – Campos da tabela ACO

ACO		
Atributo	**Tipo de Dados**	**Descrição**
NumCol	Integer	Número de colónias
NumAnts	Integer	Número de formigas por colónia
EvapRate	Double	Taxa de evaporação
Alpha	Double	Importância do valor heurístico
Beta	Double	Importância da feromona
StopCrit	Integer	Critério de paragem, número de iterações

Tabela 12 – Campos da tabela GA

GA		
Atributo	**Tipo de Dados**	**Descrição**
InitPopGen	Double	Percentagem de geração da população inicial
PopSize	Double	Percentagem do tamanho da população inicial
NumGen	Integer	Número de gerações
CrossRate	Double	Taxa de cruzamento
MutRate	Double	Taxa de mutação

Tabela 13 – Campos da tabela PSO

PSO		
Atributo	**Tipo de Dados**	**Descrição**
NumPart	Integer	Número de partículas
NumIter	Integer	Número de iterações
LowLimit	Integer	Limite inferior
UpLimit	Integer	Limite superior
MinInertia	Double	Inércia mínima
MaxInertia	Double	Inércia máxima
C1	Double	Componente cognitivo
C2	Double	Componente social
MinVeloc	Double	Velocidade mínima
MaxVeloc	Double	Velocidade máxima

Tabela 14 – Campos da tabela SA

SA		
Atributo	**Tipo de Dados**	**Descrição**
InitTemp	Double	Temperatura inicial
Alpha	Double	Fator *alpha* de redução da temperatura, ou seja, arrefecimento
NumIteraK	Integer	Número de iterações à mesma temperatura
StopCrit	Integer	Critério de paragem, número de iterações

Tabela 15 – Campos da tabela TS

TS		
Atributo	**Tipo de Dados**	**Descrição**
NeighGen	Double	Percentagem de geração de vizinhança
SubNeigh	Double	Percentagem de subvizinhança
TabuListLen	Interger	Tamanho da lista tabu
StopCrit	Integer	Critério de paragem, número de iterações

5.3.2 Algoritmo Q-Learning

O algoritmo *Q-Learning*, tem como objetivo aprender o valor do par estado-ação, $Q(s, a)$, que representa a recompensa esperada para cada par estado-ação, representados por s e a, respetivamente. Assim, para o sistema o valor ótimo do par estado-ação representa a política ótima que o aprendiz tem a intenção de aprender. Neste contexto, a Tabela 16 descreve a estrutura funcional do algoritmo.

Tabela 16 – Estrutura funcional do algoritmo Q-Learning

Algoritmo Q-Learning
1. Inicializar os valores $Q(s, a)$ aleatoriamente ou a zero 2. Selecionar o estado inicial (S_0) aleatoriamente 3. Utilizar uma política ε-greedy para selecionar a ação (a) apropriada para o estado (S_0) 4. Executar a ação (a) selecionada, receber a recompensa (r), e escolher o estado seguinte (S_1) 5. Atualizar o valor de $Q(s, a)$ da seguinte forma: $$Q(s_0, a) = Q(s_0, a) + \alpha\,[r + \gamma\, max\, Q(s_1, b) - Q(s_0, a)]$$ 6. Atualizar o estado, $S_0 = S_1$ 7. Repetir o passo 3 até que S_0 represente um estado terminal 8. Repetir os passos 2 a 7, para um determinado número de vezes

No algoritmo, cada iteração, passos 2 a 7, representa um ciclo de aprendizagem. O parâmetro, α, representa a influência da taxa de aprendizagem, enquanto o parâmetro, γ, representa a influência do valor das futuras recompensas. A matriz que relaciona o par estado-ação e armazena o resultado da função-Q é inicializada com o mesmo valor para todos os pares estado-ação, ou seja o valor 0 (zero). O passo 3 representa o *tradeoff* entre a fase de exploração e de aproveitamento.

5.3.2.1 Critério de Avaliação do Estado

As decisões tomadas por um aprendiz baseiam-se no estado atual do sistema. O estado do sistema é a base que permite a um aprendiz selecionar de um conjunto de ações a mais apropriada. Várias opções estão disponíveis para definir o estado do sistema: estas incluem medidas como o número de tarefas (*jobs*) no *buffer*, a minimização do tempo total de conclusão das tarefas, ou tempo médio do percurso. Assim, a minimização do tempo total de conclusão das tarefas foi o critério adotado para avaliar um determinado estado.

De forma a avaliar o desempenho do algoritmo, aplicado a um problema de escalonamento, este apenas é utilizado quando existem dois estados para avaliação. Onde, o estado reflete o resultado da aplicação de uma Meta-heurística a um problema de escalonamento. Assim, se houver apenas um estado, o algoritmo não atualiza o valor $Q(s,a)$, pois a decisão relativa à próxima ação é realizada sem considerar o algoritmo *Q-Learning*.

5.3.2.2 Exploração e Aproveitamento

A exploração e o aproveitamento são conceitos fundamentais quando se aplica um algoritmo de Aprendizagem por Reforço, como o *Q-Learning*. Assim, a exploração significa que o aprendiz procurar algo que não foi feito antes, de forma a obter uma maior recompensa. Por outro lado, o aproveitamento significa que o aprendiz prefere as ações que foram tomadas anteriormente recompensadas. Neste sentido, o aproveitamento pode garantir a obtenção de uma boa recompensa, no entanto, a exploração a longo prazo pode proporcionar mais oportunidades para maximizar a recompensa total. Uma abordagem comum para lidar com a questão de *tradeoff* é chamado método ε-*greedy*. O método seleciona aleatoriamente, a cada iteração, uma ação com uma probabilidade fixa, $0 \leq \varepsilon \leq 1$, em vez de selecionar de forma *greedy* uma das ações aprendidas pela função-Q. Este processo é apresentado na equação (9):

$$\tau(s) = \begin{cases} \text{ação aleatória de } \lambda(s), & \text{se } \kappa < \varepsilon \\ \text{argmax}_{\alpha \in \lambda(s)}\, Q(s,a), & \text{caso contrário} \end{cases} \qquad (9)$$

onde $0 \leq \kappa \leq 1$ é um número aleatório definido a cada nova iteração.

5.3.2.3 Função de Recompensa

A recompensa define o objetivo do aprendiz e determina o valor da ação imediata com base no estado do sistema. Uma vez que o aprendiz procura maximizar a recompensa total, a função de recompensa é utilizada essencialmente para orientar o aprendiz, para que este alcance o seu objetivo. Neste sentido, o objetivo do sistema é a minimização do tempo total da conclusão das tarefas (C_{max}). Assim, quando é gerado um novo plano de escalonamento, o C_{max} obtido é comparado com o C_{max} corrente. Se o novo C_{max} for maior que o C_{max} do estado corrente, o aprendiz recebe uma recompensa de -1. Caso contrário, o aprendiz recebe uma recompensa de 1. Por outras palavras, quando uma ação origina um C_{max} maior que a ação anterior, esta irá receber uma penalização na tabela de pares estado-ação. Neste sentido, o processo de recompensa pode ser descrito da seguinte forma:

- Recompensa de 1, se o plano de escalonamento melhorar, ou seja, o C_{max} diminuir;
- Recompensa de -1, se o plano de escalonamento piorar, ou seja, o C_{max} aumentar.

Este tipo de recompensa permite estimular o processo de aprendizagem.

5.4 Exemplo Ilustrativo

Nesta seção é apresentado um exemplo ilustrativo do funcionamento da Híper-heurística. O exemplo consiste na chegada de um novo problema ao sistema, que é resolvido pela Híper-heurística desenvolvida. Para este exemplo foram utilizadas instâncias de problemas de Lawrence (1984).

Assim, chega ao sistema um novo caso, definido na Tabela 17, que consiste num problema *Job-Shop*, com 10 tarefas e 10 máquinas, e é uma instância do problema *La05* (Lawrence, 1984).

Tabela 17 – Exemplo de um novo problema

Número de tarefas	Número de máquinas	Tipo de problema	Multi-nível	Tempo de conclusão ótimo
10	10	Job-Shop	Não	593

Neste sentido, são apresentados na Tabela 18, os registos armazenados na base de dados, onde é possível visualizar a Meta-heurística utilizada, a solução inicial e a função objetivo. Para além disso, também estão disponíveis os tempos de conclusão e de execução dos registos.

Tabela 18 – Exemplo de um conjunto de registos

ID	MH	NJobs	NMach	Problem Type	Multi Level	Cmax Optimal	InitSol	Obj Func	Cmax	Time Exec
1	ABC	15	5	Job-Shop	Não	926	SeqNivel	Cmax	60	0.15
2	SA	20	5	Job-Shop	Não	1039	SeqNivel	WT	60	0.18
3	TS	10	10	Job-Shop	Não	945	SeqNivel	Cmax	1413	12.69
4	SA	10	10	Job-Shop	Não	784	SeqNivel	WT	1325	3.55
5	TS	20	5	Job-Shop	Não	1165	SeqNivel	Cmax	1692	1.99
6	ABC	20	5	Job-Shop	Não	1150	SeqNivel	WT	1658	1.6
7	TS	10	5	Job-Shop	Não	666	SeqNivel	Cmax	753	7.64
8	TS	10	5	Job-Shop	Não	655	SeqNivel	WT	764	5.33
9	SA	10	10	Job-Shop	Não	655	SeqNivel	Cmax	839	7.57
10	ABC	10	5	Job-Shop	Não	842	SeqNivel	WT	838	7.46

Para este exemplo, considerou-se um conjunto de 10 registos, tendo estes casos sido resolvidos de forma diferenciada. O preenchimento da base de dados com os registos foi realizado da seguinte forma:

- Os registos de 1 a 4 e os de 6 a 9, são instâncias dos problemas *La01*, *La02*, *La06*, *La12*, *La13*, *La16*, *La17*, *La19*;
- Os registos 5 e 10 foi introduzido com base em resultados de execuções já obtidos anteriormente.

Os parâmetros utilizados nas Meta-heurísticas encontram-se descritos na Tabela 19, Tabela 20 e Tabela 21, respetivamente para os algoritmos ABC, SA, TS.

Tabela 19 – Exemplo de parametrização ABC

ID	PopSize	MaxFailure	NumCycles
1	100	2000	4500
6	100	2000	4500
10	50	1000	3000

Tabela 20 – Exemplo de parametrização SA

ID	StopCrit	NumIteraK	InitTemp	Alpha
2	50	15	15	0.80
4	35	15	15	0.80
9	35	15	15	0.80

Tabela 21 – Exemplo de parametrização TS

ID	StopCrit	NeighGen	SubNeigh	TabuListLen
3	100	0.15	0.1	1
5	100	0.15	0.1	3
7	100	0.15	0.1	1
8	100	0.15	0.1	1

O algoritmo descrito no subcapítulo 5.3.2 é utilizado para que o aprendiz aprenda com a experiência. Cada iteração é o equivalente a um ciclo de treino. Em cada ciclo de treino, o aprendiz explora o ambiente (representado pela matriz de recompensa R) e recebe a recompensa. O objetivo do treino é melhorar o "cérebro" do aprendiz, representado pela matriz Q. Assim, se a matriz Q estiver optimizada o processo de procura por uma boa solução é muito mais eficiente.

No sentido de uma melhor compreensão sobre o algoritmo, apresenta-se de seguida o seu funcionamento. No entanto, antes do processo iniciar é realizada uma consulta à base de dados para selecionar o conjunto de registos que irá ser utilizado pelo algoritmo como espaço de pesquisa. O comando SELECT utilizado para selecionar os registos é apresentado na Tabela 22, sendo selecionados os registos com *Njobs* entre 5 e 20 tarefas, e *NMachines* entre 2 e 40 máquinas.

Tabela 22 – Comando SELECT

Comando SELECT
SELECT rl **FROM** MetaHeuristicResult rl **WHERE** (rl.nJobs between (nJobs * 0.5) and (nJobs / 0.5)) AND (rl.nMachines between (nMachines * 0.2) and (nMachines / 0.2))

Assim, com base nos limites definidos todos os registos serão selecionados e disponibilizados ao algoritmo de aprendizagem. Estes registos representam os diferentes estados utilizados no processo de aprendizagem.

Com os registos selecionados o processo de aprendizagem pode ser iniciado. Neste contexto, e para este exemplo ilustrativo, vamos assumir que o sistema foi executado duas vezes para o problema em análise, e que produziu os resultados que se encontram na Tabela 23.

Tabela 23 – Resultados produzidos pelo sistema após duas iterações

Run	CMax	TMax	N	SumFj	SumTj	SumWjCj	SumWjTj	Fmax	U	Time
1	1231	286	10	5351	1497	10947	1497	1231	43,47	1,85
2	1242	297	10	5351	2210	11660	2210	1242	43,09	0,68

As matrizes de recompensa R e valores Q, respetivamente Tabela 24 e Tabela 25, encontram-se no seguinte estado:

Tabela 24 – Matriz de recompensa

$R =$

	ABC	SA	TS
S0	0	-	-
S1	-	100	-
S2	-	-	0
S3	-	0	-
S4	-	-	0
S5	100	-	-
S6	-	-	0
S7	-	-	0
S8	-	0	-
S9	0	-	-

Tabela 25 – Matriz de valores Q

$Q =$

	ABC	SA	TS
S0	0	0	0
S1	0	100	0
S2	0	0	0
S3	0	0	0
S4	0	0	0
S5	100	0	0
S6	0	0	0
S7	0	0	0
S8	0	0	0
S9	0	0	0

Neste contexto, vamos assumir que o estado $S1$ é escolhido aleatoriamente. A este estado está associada uma ação, que neste caso é a Meta-heurística *Simulated Annealing.* Assim, a parametrização associada à Meta-heurística é enviada para o sistema e é gerado o resultado apresentado na Tabela 26.

Tabela 26 – Resultado gerado após a aplicação da Meta-heurística

Run	CMax	TMax	N	SumFj	SumTj	SumWjCj	SumWjTj	Fmax	U	Time
1	1231	286	10	5351	1497	10947	1497	1231	43,47	1,85
2	1242	297	10	5351	2210	11660	2210	1242	43,09	0,68
3	1120	280	10	5351	1394	10847	1397	1234	43.4	2.28

O resultado é avaliado de forma a ser atribuída a devida recompensa. O critério utilizado para avaliar o estado é a minimização do C_{max}, ou seja, se o C_{max} gerado for inferior ao anterior então é atribuída uma recompensa positiva. Como foi o caso, o estado será atualizado na matriz de recompensas com a adição de mais 1 ao valor existente. Seguidamente, o valor da matriz Q é atualizado através da equação (10):

$$Q(s,a) = Q(s,a) + \alpha\,[r + \gamma\, max\, Q(s',a') - Q(s,a)] \qquad (10)$$

Assim, assumindo que o valor de $\alpha = 0.1$ e de $\gamma = 0.9$, o resultado é o seguinte:

$$Q(s1,SA) = 100 + 0.1\,[101 + 0.9 * 100 - 100] = 109.1 \qquad (11)$$

As matrizes de recompensa R e valores Q, respetivamente Tabela 27 e Tabela 28, após a atribuída a recompensa e atualizado o valor de Q encontram-se no seguinte estado:

Tabela 27 – Matriz de recompensa

$R =$

	ABC	SA	TS
S0	0	-	-
S1	-	101	-
S2	-	-	0
S3	-	0	-
S4	-	-	0
S5	100	-	-
S6	-	-	0
S7	-	-	0
S8	-	0	-
S9	0	-	-

Tabela 28 – Matriz de valores Q

$Q =$

	ABC	SA	TS
S0	0	0	0
S1	0	109.1	0
S2	0	0	0
S3	0	0	0
S4	0	0	0
S5	100	0	0
S6	0	0	0
S7	0	0	0
S8	0	0	0
S9	0	0	0

Finalmente, o resultado gerado é avaliado de forma a validar se o valor obtido é inferior ao anterior, como é caso o resultado é armazenado na base de dados para futura utilização, como demonstrado na Tabela 29. Bem como a parametrização associada à Meta-heurística que gerou a solução, como demonstrado na Tabela 30.

Tabela 29 – Resultado inserido na base de dados

ID	MH	NJobs	NMach	Problem Type	Multi Level	Cmax Optimal	InitSol	Obj Func	Cmax	Time Exec
1	ABC	15	5	Job-Shop	Não	926	SeqNivel	Cmax	60	0.15
2	SA	20	5	Job-Shop	Não	1039	SeqNivel	WT	60	0.18
3	TS	10	10	Job-Shop	Não	945	SeqNivel	Cmax	1413	12.69
4	SA	10	10	Job-Shop	Não	784	SeqNivel	WT	1325	3.55
5	TS	20	5	Job-Shop	Não	1165	SeqNivel	Cmax	1692	1.99
6	ABC	20	5	Job-Shop	Não	1150	SeqNivel	WT	1658	1.6
7	TS	10	5	Job-Shop	Não	666	SeqNivel	Cmax	753	7.64
8	TS	10	5	Job-Shop	Não	655	SeqNivel	WT	764	5.33
9	SA	10	10	Job-Shop	Não	655	SeqNivel	Cmax	839	7.57
10	ABC	10	5	Job-Shop	Não	842	SeqNivel	WT	838	7.46
11	SA	10	10	Job-Shop	Não	593	SeqNivel	WT	1120	2.28

Tabela 30 – Parametrização final do SA

ID	StopCrit	NumIteraK	InitTemp	Alpha
2	50	15	15	0.80
4	35	15	15	0.80
9	35	15	15	0.80
11	35	15	15	0.80

5.5 Sumário do Capítulo

Neste capítulo foi descrito o sistema AutoDynAgents, que consiste num Sistema Multiagente para a resolução de problemas de escalonamento sujeitos a perturbações. Assim, foi feita uma descrição do modelo de agentes no qual assenta a arquitetura do sistema (secção 5.2), bem como de alguns módulos, nomeadamente, módulo de escalonamento e do mecanismo de coordenação (secção 5.2.1.3).

Sendo o âmbito desta tese, foi dada enfâse à Híper-heurística (secção 5.3) e à forma como este módulo interage com o sistema AutoDynAgents (secção 5.2.1). O desenvolvimento deste módulo surge da necessidade de dotar o sistema de capacidades de aprendizagem, através do uso de experiência. Assim, para resolução do problema foi proposta uma Híper-heurística que incorpora um método de Aprendizagem por Reforço (subcapítulo 3.2.3), mais propriamente o algoritmo *Q-Learning*, descrito na secção 3.3.3. A Híper-heurística é capaz de selecionar e configurar os parâmetros das diferentes Meta-heurísticas, para a resolução dos diferentes problemas que vão surgindo no sistema. Este processo foi demonstrado num exemplo ilustrativo na secção 5.4.

Capítulo 6

Estudo Computacional

Neste capítulo, é apresentado e descrito o conjunto de teste computacionais realizados, com o objetivo de validar o desempenho da Híper-heurística proposta. Serão apresentados os resultados computacionais obtidos no âmbito deste trabalho, tendo sido utilizadas as instâncias de problemas de testes académicos (*Job-Shop*) de Lawrence (1984), Adams et al. (1988), Applegate e Cook (1991) e Storer et al. (1992a).

A implementação da Híper-heurística foi realizada na linguagem de programação Java. O acesso à Base de Dados foi efetuado através da *framework Hibernate*[7], que permite o mapeamento entre um modelo de programação orientada a objetos e bases de dados relacionais. A base de dados foi desenvolvida em H2[8], sendo este um sistema de gestão de bases de dados relacionais.

A máquina utilizada no estudo computacional tem as seguintes características: processador Intel Core 2 T5500 @ 1.66 GHz, 2GB de memoria RAM, disco rígido Samsung de 500GB, com o sistema operativo Windows 7 Profissional.

O estudo computacional encontra-se dividido em duas partes. Na primeira parte, apresenta-se o processo de inicialização da Base de Dados, onde para cada instância de problemas, foram inseridos os melhores resultados obtidos pelo sistema AutoDynAgents, utilizando as diferentes Meta-heurísticas implementadas e para cada uma delas as seguintes medidas de otimização (função objetivo): C_{max} (tempo total de conclusão) e WT (soma dos atrasos pesados). As Meta-heurísticas utilizadas foram, tal como apresentadas e descritas na secção 4.2, os Algoritmos Genéticos, a Colónia Artificial de Abelhas, a Otimização por Colónia de Formigas, o *Particle Swarm Optimization*, a Pesquisa Tabu e o *Simulated Annealing* e as respetivas parametrizações encontram-se definidas na secção 6.1.2. É de salientar que o resultado obtido para cada Meta-heurística, independentemente da função objetivo utilizada, é avaliado segundo o tempo de conclusão (C_{max}) e tempo de execução (*TimeExec*), tal como apresentado nas tabelas de resultados computacionais.

Na segunda parte, apresentam-se os resultados computacionais obtidos, bem como os resultados utilizados como método de comparação, onde se incluem os resultados previamente obtidos pelo AutoDynAgents, ou seja, sem a aplicação de técnicas de aprendizagem. A notação utilizada nas tabelas de resultados encontra-se descrita na Tabela 31.

[7] www.hibernate.org

[8] www.h2database.com

Tabela 31 – Notação das tabelas de resultados computacionais

Campo	Descrição
Prob.	Instância do problema
n	Número de tarefas
m	Número de máquinas
C_{max} Ótimo	Tempo de conclusão ótimo
Func. Obj.	Critério de otimização ou função objetivo
C_{max}	Tempo total de conclusão
WT	Soma dos atrasos pesados (Weighted Tardiness)
TimeExec (s)	Tempo de execução em segundos

6.1 Inicialização da Base de Dados

Numa fase inicial, foi necessário inicializar a Base de Dados com os melhores resultados obtidos anteriormente pelo sistema AutoDynAgents. Este processo de inicialização tem como objetivo, criar um ponto de partida para a evolução da Híper-heurística.

Os resultados inseridos, como mencionado, correspondem aos melhores resultados obtidos pelo sistema AutoDynAgents com as diferentes Meta-heurísticas, paras as funções objetivos, C_{max} e WT. As parametrizações utilizadas nas Meta-heurísticas correspondem ao resultado de algum conhecimento pericial e experiências de tentativa-erro.

6.1.1 Resultados Inseridos na Base de Dados

Na Tabela 32 são apresentados os melhores resultados obtidos para um conjunto de 30 instâncias de problemas de testes académicos (*Job-Shop*) de Lawrence (1984), Adams et al. (1988), Applegate e Cook (1991), Storer et al. (1992a), Taillard (1993) e Yamada e Nakano (1992). Estes resultados consistem nos melhores tempos de conclusão (C_{max}) e tempos de execução (*TimeExec*) obtidos com as Meta-heurísticas Algoritmos Genéticos, Otimização por Colónia de Formigas, *Particle Swarm Optimization*, Pesquisa Tabu e *Simulated Annealing.*

Os resultados obtidos pelas Meta-heurísticas Algoritmos Genéticos, Otimização por Colónia de Formigas, *Particle Swarm Optimization*, Pesquisa Tabu e *Simulated Annealing*, encontram-se publicados em (Pereira e Madureira, 2013).

Tabela 32 – Tempos de conclusão e execução das Meta-heurísticas

Prob.	*n*	*m*	F. Obj.	TS		GA		SA		ACO		PSO	
				C_{max}	Time Exec	C_{max}	Time Exec	C_{max}	Time Exec	C_{max}	Time Exec	C_{max}	Time Exec
La02	10	5	C_{max}	839	7,57	839	10,7	877	0,07	865	0,31	923	0,69
	10	5	WT	838	7,46	837	10,6	839	0,14	853	0,43	852	0,71
La04	10	5	C_{max}	796	5,61	799	11,2	924	0,08	909	0,36	810	0,69
	10	5	WT	755	6,03	753	112	765	0,12	855	0,49	921	0,71
La07	15	5	C_{max}	1037	0,75	1150	27,3	1020	0,15	1170	5,8	1049	2,61
	15	5	WT	1031	0,84	1020	27,4	1039	0,2	1025	7,07	1045	2,75
La08	15	5	C_{max}	1014	0,74	1049	31,9	1005	0,14	1009	5,77	1018	2,72
	15	5	WT	981	0,77	1009	32	975	0,21	1021	5,17	958	2,44
La09	15	5	C_{max}	966	0,78	1024	31,9	951	0,18	994	5,86	979	2,72
	15	5	WT	951	0,77	951	32	951	0,21	951	5,18	951	1,64
La13	20	5	C_{max}	1168	1,46	1183	55,2	1158	0,22	1190	11,94	1205	4
	20	5	WT	1170	1,36	1201	55,1	1150	0,25	1164	14,72	1150	4,07
La16	10	10	C_{max}	1279	2,85	1213	9,06	1549	0,07	2137	0,1	1516	1,26
	10	10	WT	1144	2,94	1326	7,99	1144	0,27	1210	0,82	1144	1,38
La20	10	10	C_{max}	1364	2,61	1439	8,57	1693	0,1	2059	0,11	1396	0,1
	10	10	WT	1185	2,69	1322	8,17	1156	0,27	1159	0,83	1090	1,16
La22	15	10	C_{max}	1466	1,5	1321	28,8	1717	0,07	1627	0,15	1622	3,37
	15	10	WT	1293	1,59	1462	28,9	1334	0,33	1378	8,61	1339	5,09
La23	15	10	C_{max}	1471	1,6	1464	29,8	1563	0,21	1361	10,31	1307	3,45
	15	10	WT	1334	1,86	1426	29,7	1345	0,36	1340	12,98	1366	4,62
La26	20	10	C_{max}	1664	2,31	1708	53,8	1665	0,32	1644	20,52	1687	6,74
	20	10	WT	1603	3,33	1627	52,4	1657	0,32	1660	25,26	1671	6,72
La32	30	10	C_{max}	2247	15,99	2244	168	2533	3,9	2200	71,18	2237	37,19
	30	10	WT	2224	18,07	2365	177	2201	4,62	2366	71,96	2121	18,15
La34	30	10	C_{max}	2160	13,39	2123	156	2135	4,43	2129	45,97	2087	34,47
	30	10	WT	2067	11,57	2016	156	2207	4,15	2069	83,04	2156	34,46
La38	15	15	C_{max}	1842	2,09	1926	18	2439	0,16	2714	0,32	2835	0,11
	15	15	WT	1696	2,53	1783	18,6	1658	0,4	1686	19,56	1705	5,79
ABZ6	10	10	C_{max}	1179	1,99	1348	11	1444	0,19	1632	0,11	1323	1,18
	10	10	WT	1095	5,11	1264	11	1164	0,21	1265	1,24	1128	1,41
ABZ8	20	15	C_{max}	986	4,93	985	58	1391	0,25	988	22,63	1111	9,32
	20	15	WT	885	4,28	1018	59	913	0,71	1009	25,51	970	10,12
ORB1	10	10	C_{max}	1611	16,71	1618	11,2	1933	0,18	1857	0,71	1720	0,91
	10	10	WT	1480	15,18	1695	11,3	1448	0,21	1605	0,77	1551	1,39
SWV1	20	10	C_{max}	2648	4,24	2540	63,5	2793	0,13	3499	0,31	2711	0,13
	20	10	WT	2460	0,3	2481	63,9	2634	0,44	3242	0,32	2506	0,3

SWV8	20	15	C_{max}	3153	5,31	2998	53,6	3448	0,19	4256	0,35	3481	0,35
	20	15	WT	3008	5,43	2993	54	3181	0,27	3660	0,43	3047	7,51
SWV 12	50	10	C_{max}	5276	81,04	5205	420	5654	0,23	5916	346,4	5790	0,33
	50	10	WT	5445	76,13	5094	429	6373	0,21	7895	1,72	6308	0,21
SWV 13	50	10	C_{max}	5622	86,92	5617	409	6056	0,19	6008	1,5	5791	136,27
	50	10	WT	5654	93,1	5600	422	6075	6,1	7736	462,5	5637	136,85
SWV 14	50	10	C_{max}	5339	74,71	5430	396	5561	0,29	5913	1,52	5951	133,63
	50	10	WT	5128	76,25	5277	417	4891	6,42	5632	469,5	5317	125,14
SWV 16	50	10	C_{max}	3098	88,47	3192	399	3120	5,52	4518	461,8	3211	40,96
	50	10	WT	3157	72,58	3122	407	3120	6,82	3178	582,5	3258	40,6
SWV 17	50	10	C_{max}	3280	78,36	3339	458	3167	5,97	3252	470,2	3310	56,97
	50	10	WT	3238	75,24	3367	445	3338	6,36	3340	602,2	3239	116,57
SWV 18	50	10	C_{max}	3000	91,36	2939	393	3016	5,67	3042	349,8	3127	139,5
	50	10	WT	2910	72,86	2991	393	2960	6,41	2975	491,3	3049	138,18
SWV 19	50	10	C_{max}	3413	77,76	3621	431	3603	6,45	3651	350,2	3699	40,42
	50	10	WT	3616	101,2	3567	424	3575	6,27	3572	457,2	3458	44,04
SWV 20	50	10	C_{max}	2955	86,64	3082	383	2997	6,14	2933	356,6	3029	138
	50	10	WT	2943	71,43	2886	403	2926	6,42	2944	445,3	2943	139,06
YN1	20	20	C_{max}	1464	5,77	1380	35	1882	0,27	1606	6,51	1575	2,31
	20	20	WT	1260	5,97	1380	36	1240	0,86	1242	44,57	1184	9,82
YN2	20	20	C_{max}	1542	0,41	1403	45	1804	0,19	2059	0,56	1853	0,29
	20	20	WT	1235	5,97	1356	45	1216	0,97	1252	46,7	1237	12,14
YN3	20	20	C_{max}	1638	6,31	1386	60	1708	0,26	1382	26,06	1484	0,3
	20	20	WT	1242	4,88	1515	60	1289	0,5	1375	44,23	1287	12,15

Na Tabela 33 são apresentados os melhores resultados obtidos para um conjunto de instâncias de problemas de testes académicos (*Job-Shop*) de Lawrence (1984), Adams et al. (1988), Applegate e Cook (1991), Storer et al. (1992a), Taillard (1993) e Yamada e Nakano (1992). Estes resultados consistem nos melhores tempos de conclusão (C_{max}) e tempos de execução (*TimeExec*) obtidos com a Meta-heurística Colónia Artificial de Abelhas.

Refira-se que a separação de resultados obtidos deve-se ao facto de as instâncias utilizadas para a Colónia Artificial de Abelhas não serem as mesmas utilizadas para as restantes Meta-heurísticas, pois as instâncias onde a Colónia Artificial de Abelhas obteve melhores resultados são diferentes. Como, o objetivo é carregar a base de dados com os melhores resultados obtidos pelo sistema AutoDynAgents para as diferentes Meta-heurísticas, optou-se por realizar esta separação e assim garantir que os resultados utilizados para a Colónia Artificial de Abelhas são efetivamente os melhores.

Tabela 33 – Tempos de conclusão e execução para a Meta-heurística ABC

Prob.	*n*	*m*	F. Obj.	ABC	
				C_{max}	Time Exec
La02	10	5	C_{max}	951	0,54
	10	5	WT	858	0,10
La04	10	5	C_{max}	821	0,22
	10	5	WT	847	0,13
La05	10	5	C_{max}	603	0,30
La07	15	5	WT	1097	0,16
La06	15	5	C_{max}	998	0,24
La08	15	5	WT	963	0,29
La09	15	5	WT	1000	0,26
La11	20	5	C_{max}	1285	0,32
La13	20	5	WT	1243	0,27
La16	10	10	C_{max}	1231	0,49
	10	10	WT	1144	0,15
La17	10	10	C_{max}	925	0,35
La20	10	10	WT	1090	0,16
La21	15	10	C_{max}	1367	0,22
La22	15	10	WT	1286	0,15
La23	15	10	WT	1256	0,17
La26	20	10	WT	1599	0,15
La29	20	10	C_{max}	1617	0,32
La31	30	10	C_{max}	2311	0,25
La32	30	10	WT	2240	0,22
La34	30	10	WT	2133	0,15
La35	30	10	C_{max}	2513	0,43
La38	15	15	WT	1668	0,16
La40	15	15	C_{max}	1729	0,27
ABZ5	10	10	C_{max}	1682	0,43
ABZ6	10	10	C_{max}	1152	0,27
	10	10	WT	1087	0,18
ABZ7	20	15	C_{max}	917	0,43
ABZ8	20	15	C_{max}	999	0,40
	20	15	WT	939	0,22
ABZ9	20	15	C_{max}	1022	0,52
ORB1	10	10	C_{max}	1612	0,65
	10	10	WT	1481	0,27
ORB3	10	10	C_{max}	1725	0,38
ORB5	10	10	C_{max}	1469	0,46

SWV1	20	10	WT	2902	0,25
SWV4	20	10	C_{max}	2600	0,52
SWV5	20	10	C_{max}	2495	0,32
SWV6	20	15	C_{max}	2690	0,49
SWV8	20	15	WT	3156	0,23
SWV9	20	15	C_{max}	2961	0,53
SWV11	50	10	C_{max}	5284	0,49
SWV 12	50	10	WT	6172	0,26
SWV 13	50	10	WT	6105	0,23
SWV 14	50	10	WT	5300	0,24
SWV15	50	10	C_{max}	5607	0,27
SWV 16	50	10	WT	3152	0,22
TAI1	30	15	C_{max}	2432	0,55
TAI2	30	15	C_{max}	2952	0,29
	30	20	WT	2550	0,22
TAI4	30	20	WT	3025	0,41
TAI5	50	15	C_{max}	3868	0,55
TAI6	50	15	WT	3676	0,24
TAI8	50	20	WT	4180	0,27
TAI11	30	15	WT	2517	0,27
TAI12	30	20	WT	2921	0,29
YN2	20	20	C_{max}	1238	5,48
	20	20	WT	1246	0,28

Com base na Tabela 32 e Tabela 33, é possível verificar que foram inseridos 360 registos na Base de Dados, que corresponde à totalidade de execuções (30 instâncias de problemas x 6 Meta-heurísticas x 2 funções objetivos).

6.1.2 Parametrização Inicial

Apresenta-se de seguida para cada instância a parametrização inicial das Meta-heurísticas utilizadas na inicialização da Base de Dados. Assim, para todas a Meta-heurísticas, o algoritmo de geração da solução inicial utilizado foi o *SeqNivel* (Madureira, 2003), em que cada solução é definida pela ordenação não-decrescente dos níveis a que pertencem as operações, dando assim prioridade às operações das diferentes tarefas que devem ser processadas primeiro.

Nos Algoritmos Genéticos e Pesquisa Tabu foi utilizada 100% da vizinhança/população em todas as instâncias, assim como um critério de paragem/gerações de 100. O afastamento máximo e o tamanho da lista tabu foram variáveis de acordo com o problema, ou seja, estes valores variam em função do número de tarefas da instância em análise. Em relação às taxas de cruzamento e de mutação, foram definidas como 75% e 1%, respetivamente.

Nas restantes Meta-heurísticas foi utilizado um mecanismo de geração de vizinhança de troca de operações por níveis, que consiste na troca de operações pertencentes ao mesmo nível de gama operatória de modo a evitar a geração de soluções impossíveis (Madureira, 2003).

No *Simulated Annealing*, a temperatura foi inicializada a 15° com um fator de arrefecimento de 80% para todos os problemas. O número de iterações à mesma temperatura e o critério de paragem variam proporcionalmente à dimensão da instância em análise. Na Otimização por Colónia de Formigas varia o número de formigas e o critério de paragem, sendo o número de colónias 1, a taxa de evaporação da feromona de 80% e os critérios *Alpha* e *Beta* 1. O *Particle Swarm Optimization* só varia no número de partículas e no número de iterações de acordo com a instância.

Na Colónia Artificial de Abelhas foi utilizado nas instâncias *La02* e *La04* um número de população de 50, assim como um critério de paragem de 3000. Para além disso, a falha máxima, parâmetro que define a distância que a abelha pode percorrer antes de ter que voltar à colónia, foi definida com um valor de 1000. Nas restantes instâncias foi utilizado um número de população de 100, um critério de paragem de 4500 e um número de falha máxima de 2000.

A parametrização das Meta-heurísticas Algoritmos Genéticos, Otimização por Colónia de Formigas, *Particle Swarm Optimization*, Pesquisa Tabu, *Simulated Annealing* e Colónia Artificial de Abelhas encontra-se descrita na Tabela 34, Tabela 35, Tabela 36 e Tabela 37, respetivamente.

Tabela 34 – Parametrizações para instâncias de 10 tarefas

Meta-heurística	Parâmetro	Valor
Algoritmos Genéticos	Número de gerações	100
	Afastamento	15%
	Tamanho da população	100%
	Taxa de cruzamento	75%
	Taxa de mutação	1%
Otimização por Colónia de Formigas	Critério de paragem	100
	Taxa de evaporação	80%
	Número de formigas por colónia	25
	Número de colónias	1
	Alpha	1
	Beta	1
Particle Swarm Optimization	Número de iterações	1000
	Número de partículas	25
	Velocidade mínima	-4
	Velocidade máxima	4
	Inércia mínima	40%
	Inércia máxima	95%
	C1	2,0

	C2	2,0
	Limite inferior	0
	Limite superior	4
Pesquisa Tabu	Critério de paragem	100
	Afastamento	15%
	Subvizinhança	100%
	Tamanho da Lista Tabu	1
Simulated Annealing	Critério de paragem	35
	Número de iterações à mesma temperatura	15
	Temperatura inicial	15
	Factor de redução da temperatura (*alpha*)	80%

Tabela 35 – Parametrizações para instâncias de 20 e 15 tarefas

Meta-heurística	Parâmetro	Valor
Algoritmos Genéticos	Número de gerações	100
	Afastamento	15%
	Tamanho da população	100%
	Taxa de cruzamento	75%
	Taxa de mutação	1%
Otimização por Colónia de Formigas	Critério de paragem	250
	Taxa de evaporação	80%
	Número de formigas por colónia	50
	Número de colónias	1
	Alpha	1
	Beta	1
Particle Swarm Optimization	Número de iterações	2000
	Número de partículas	35
	Velocidade mínima	-4
	Velocidade máxima	4
	Inércia mínima	40%
	Inércia máxima	95%
	C1	2,0
	C2	2,0
	Limite inferior	0
	Limite superior	4
Pesquisa Tabu	Critério de paragem	100
	Afastamento	15%
	Subvizinhança	100%
	Tamanho da Lista Tabu	3

Simulated Annealing	Critério de paragem	50
	Número de iterações à mesma temperatura	15
	Temperatura inicial	15
	Factor de redução da temperatura (*alpha*)	80%

Tabela 36 – Parametrizações para instâncias de 30 e 50 tarefas

Meta-heurística	Parâmetro	Valor
Algoritmos Genéticos	Número de gerações	200
	Afastamento	15%
	Tamanho da população	100%
	Taxa de cruzamento	75%
	Taxa de mutação	1%
Otimização por Colónia de Formigas	Critério de paragem	300
	Taxa de evaporação	80%
	Número de formigas por colónia	75
	Número de colónias	1
	Alpha	1
	Beta	1
Particle Swarm Optimization	Número de iterações	3000
	Número de partículas	75
	Velocidade mínima	-4
	Velocidade máxima	4
	Inércia mínima	40%
	Inércia máxima	95%
	C1	2,0
	C2	2,0
	Limite inferior	0
	Limite superior	4
Pesquisa Tabu	Critério de paragem	200
	Afastamento	15%
	Subvizinhança	100%
	Tamanho da Lista Tabu	3
Simulated Annealing	Critério de paragem	100
	Número de iterações à mesma temperatura	25
	Temperatura inicial	15
	Fator de redução da temperatura (*alpha*)	80%

Tabela 37 – Parametrizações para a Meta-heurística ABC

Meta-heurística	Instâncias	Parâmetro		
		Tamanho da população	Falha máxima	Número de iterações
Colónia Artificial de Abelhas	La02	50	1000	3000
	La04	50	1000	3000
	Restante instâncias	100	2000	4500

Assim, foram introduzidas, nas tabelas (ABC, ACO, GA, SA, TS, PSO) descritas na secção 5.3.1, as parametrizações para os 360 registos já inseridos na Base de Dados.

6.2 Resultados Computacionais

Nesta secção, são apresentados os resultados computacionais obtidos. Inicialmente apresentam-se os resultados obtidos pela Híper-heurística e seguidamente os resultados que servem como base de comparação. Todos os resultados apresentados correspondem à minimização do tempo de conclusão (C_{max}).

6.2.1 Resultados Obtidos com a Híper-heurística

Os resultados apresentados referem-se aos resultados obtidos pelo sistema AutoDynAgents com a Híper-heurística incorporada. Assim, para as instâncias em análise o critério de paragem do algoritmo *Q-Learning* variou entre 10, 20 e 30 corridas. Para cada uma das corridas foram recolhidos os dados e calculadas as médias dos valores obtidos. Para além disso, os valores foram normalizados, através do calculo do quociente entre o valor ótimo o valor médio do C_{max} (equação (12)), de forma a ser possível estimar o desvio entre o valor obtido e o valor da solução ótima referenciada na literatura.

$$q = 1 - \frac{\mathrm{OptCmax}}{\mathrm{Cmax}} \qquad (12)$$

Assim, em vez de uma comparação direta entre valores obtidos, compara-se a variação do valor médio de C_{max} em relação ao ótimo, o que se revela proveitoso quando os valores de C_{max} são diferentes. Por exemplo, duas instâncias, cujos valores são respetivamente 100 e 200. Se for obtido o valor 150 para a primeira e 210 para a segunda, é possível concluir que foi obtido um valor mais próximo do ótimo na segunda, pois o quociente da primeiro é 0,333 e da segunda 0,048. Neste sentido, apresenta-se na Tabela 38 os valores médios devidamente normalizados.

Tabela 38 – Resultados computacionais obtidos para um critério de paragem de 10, 20 e 30

Prob.	*n*	*m*	C_{max} Ótimo	Critério de Paragem		
				10 Corridas	20 Corridas	30 Corridas
				C_{max}	C_{max}	C_{max}
La01	10	5	666	0,207	0,190	0,173
La02	10	5	655	0,282	0,295	0,275
La05	10	5	593	0,080	0,086	0,094
La06	15	5	926	0,102	0,082	0,078
La07	15	5	890	0,212	0,222	0,198
La08	15	5	863	0,135	0,153	0,160
La09	15	5	951	0,058	0,051	0,044
La10	15	5	958	0,016	0,047	0,027
La11	20	5	1222	0,094	0,079	0,089
La12	20	5	1039	0,149	0,161	0,174
La13	20	5	1150	0,070	0,084	0,039
La14	20	5	1292	0,008	0,009	0,010
La15	20	5	1207	0,232	0,230	0,236
La16	10	10	945	0,289	0,241	0,268
La17	10	10	784	0,188	0,218	0,211
La18	10	10	848	0,243	0,229	0,237
La19	10	10	842	0,255	0,272	0,295
La23	15	10	1032	0,305	0,305	0,314
La24	15	10	935	0,269	0,261	0,278
La31	30	10	1784	0,219	0,200	0,199
La32	30	10	1850	0,208	0,196	0,203
La33	30	10	1719	0,197	0,195	0,196
La34	30	10	1721	0,254	0,237	0,270
ABZ5	10	10	1234	0,256	0,249	0,249
ABZ6	10	10	943	0,320	0,275	0,276
ORB4	10	10	1005	0,320	0,310	0,276
SWV16	50	10	2924	0,103	0,093	0,092
SWV17	50	10	2794	0,164	0,164	0,166
SWV18	50	10	2852	0,075	0,075	0,064
SWV20	50	10	2823	0,061	0,061	0,056

Para ser possível analisar os resultados obtidos, utiliza-se o teste *t* de Student (Box, 1987). Assim, as amostras foram normalizadas para ser possível comparar as abordagens diretamente de uma forma global. Esta normalização foi realizada através do cálculo do quociente dos valores médios (Figura 15).

Da análise do boxplot (Figura 15) é possível verificar a vantagem do critério de paragem de 30 corridas, o que permite concluir acerca da obtenção de melhores resultados quando executadas as 30 corridas.

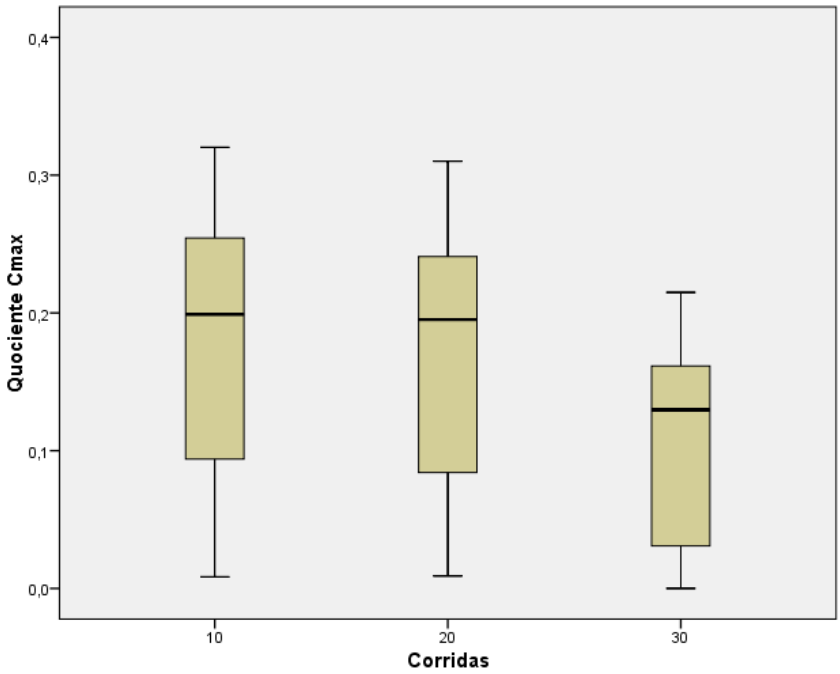

Figura 15 – Comparação do quociente dos valores médios da execução do sistema com a Híper-heurística

Com base no Teorema do Limite Central, é possível assumir a normalidade da amostra para um tamanho da amostra $\mathrm{n} \geq 30$. Utiliza-se o teste *t* de Student (Box, 1987), considerado como um teste poderoso para análise de amostras (Conover, 1999). Analisando a significância estatística destes resultados (Tabela 39), através da observação dos valores $t(29) = -0{,}181; p > 0{,}05$, verifica-se que não existem diferenças estatisticamente significativas entre os resultados obtidos nas 20 corridas face aos obtidos nas 10 corridas.

Tabela 39 – Resultado do teste *t* de Student para amostras emparelhadas: 10 Corridas *vs.* 20 Corridas, 10 Corridas *vs.* 30 Corridas, 20 Corridas *vs.* 30 Corridas

	Média da diferença	Desvio padrão da diferença	*t*	Graus de liberdade	*p_value*
10 Corridas *vs.* 20 Corridas	-0,00062	0,01891	-0,181	29	0,858
10 Corridas *vs.* 30 Corridas	0,07068	0,03963	9,768	29	0,000
20 Corridas *vs.* 30 Corridas	0,07131	0,04091	9,547	29	0,000

Por outro lado, comparando os grupos entre as 30 e 10 corridas, e observando os valores $t(29) = 9{,}768; p < 0{,}05$, pode-se afirmar, com um grau de confiança de 95%, que existem

diferenças estatisticamente significativas entre os dois resultados, com vantagem para as 30 corridas. Finalmente, comparando os resultados entre as 30 e 20 corridas, e observando os valores $t(29) = 9{,}547; p < 0{,}05$, pode-se afirmar, com um grau de confiança de 95%, que existem diferenças estatisticamente significativas entre os dois resultados, com vantagem para as 30 corridas.

Torna-se possível concluir que para um critério de paragem de 30 corridas foi possível obter melhores resultados globais e como tal, estes resultados serão utilizados para validar o desempenho da Híper-heurística, face aos resultados obtidos com e sem mecanismo aprendizagem. Para além disso, também é possível verificar que a Híper-heurística evolui favoravelmente com o aumento do número de iterações o que poderá indicar uma melhoria dos resultados com um aumento do número de iterações, por exemplo para 50. No entanto, será necessária a realização de testes computacionais adicionais para validar esta hipótese.

6.2.2 Comparação com Resultados Prévios

Na Tabela 40 são apresentados os resultados prévios, obtidos antes da implementação da Híper-heurística, que servem de comparação com os resultados obtidos pela Híper-heurística implementada no sistema AutoDynAgents (Madureira e Pereira, 2010) (Madureira et al., 2011a).

Os resultados foram obtidos após cinco corridas para cada instância considerada, utilizando-se os critérios de otimização já referidos, sendo apresentado o valor médio devidamente normalizado, como referido anteriormente.

Tabela 40 – Resultados obtidos sem mecanismo de aprendizagem

Prob.	*n*	*m*	C_{max} Ótimo	Resultados Prévios
				C_{max}
La01	10	5	666	0,264
La02	10	5	655	0,303
La05	10	5	593	0,118
La06	15	5	926	0,118
La07	15	5	890	0,220
La08	15	5	863	0,191
La09	15	5	951	0,088
La10	15	5	958	0,040
La11	20	5	1222	0,071
La12	20	5	1039	0,128
La13	20	5	1150	0,067
La14	20	5	1292	0,017
La15	20	5	1207	0,243
La16	10	10	945	0,365

La17	10	10	784	0,353
La18	10	10	848	0,362
La19	10	10	842	0,409
La23	15	10	1032	0,329
La24	15	10	935	0,393
La31	30	10	1784	0,220
La32	30	10	1850	0,253
La33	30	10	1719	0,246
La34	30	10	1721	0,249
ABZ5	10	10	1234	0,282
ABZ6	10	10	943	0,363
ORB4	10	10	1005	0,388
SWV16	50	10	2924	0,138
SWV17	50	10	2794	0,172
SWV18	50	10	2852	0,109
SWV20	50	10	2823	0,091

Assim, comparando os resultados obtidos é possível verificar, pela análise do boxplot da Figura 16 que a Híper-heurística (*Q-Learning*) para o critério de paragem de 30 corridas obteve o melhor desempenho na minimização do C_{max}. Tal é possível quer pela análise das medidas de posição (mediana e média), quer pelas medidas de dispersão (desvio padrão, variância).

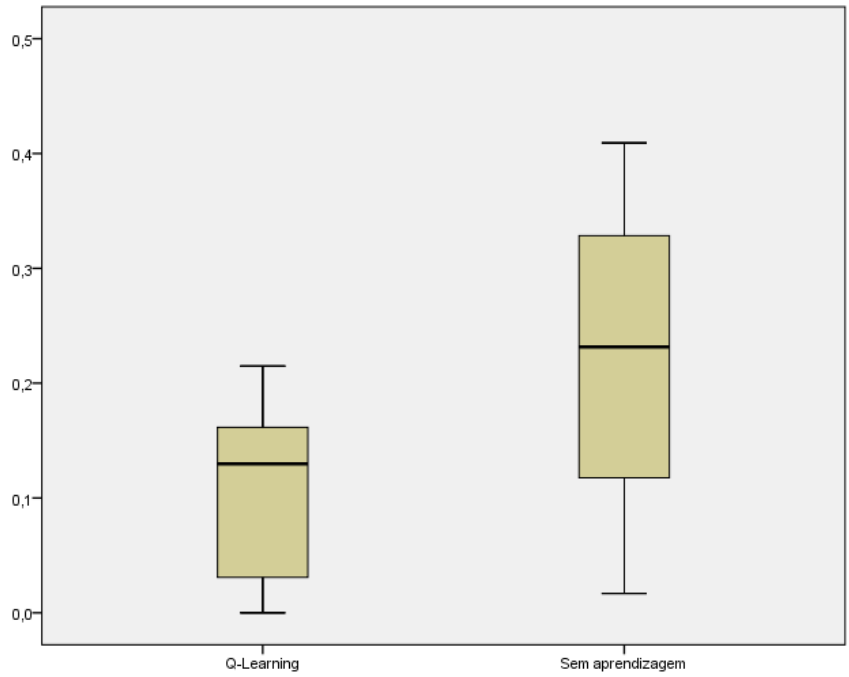

Figura 16 - Comparação do quociente dos valores médios da execução do sistema, entre os resultados prévios e os resultados obtidos pela Híper-heurística

Apesar de se verificar a existência estatística da vantagem da Híper-heurística foi usado o teste *t* de Student para analisar a significância estatística dos resultados, através da observação dos valores $t(29) = -9{,}152; p < 0{,}05$ na Tabela 41, pode-se afirmar, com um grau de confiança de 95%, que existem diferenças estatisticamente significativas entres os resultados prévios e os resultados obtidos pela Híper-heurística (30 corridas), permitindo concluir sobre a vantagem da Híper-heurística.

Tabela 41 – Resultado do teste *t* de Student para amostras emparelhadas: Híper-heurística *vs.* Resultados Prévios

	Média da diferença	Desvio padrão da diferença	*t*	Graus de liberdade	*p_value*
Híper-heurística *vs.* Resultados Prévios	-0,11529	0,06900	-9,152	29	0,000

A Híper-heurística apresentou vantagens relativamente aos resultados prévios, com a obtenção de melhores resultados médios, permitindo concluir quanto à existência de vantagem estatisticamente significativa da abordagem baseada na Híper-heurística no desempenho do sistema AutoDynAgents.

6.2.3 Comparação com a Técnica de Aprendizagem Raciocínio Baseado em Casos

Na Tabela 42 são apresentados os resultados do módulo de Raciocínio Baseado em Casos, que servem de comparação com os resultados obtidos pela Híper-heurística implementada no sistema AutoDynAgents.

O Raciocínio Baseado em Casos (*Case Based Reasoning* – CBR) é uma técnica que procura resolver novos casos/problemas adaptando soluções já utilizadas na resolução de outros problemas (Pereira, 2009). As características de funcionamento do sistema CBR são:

- Extração do conhecimento a partir de casos ou experiências que o próprio sistema encontra;
- Identificação das características mais significantes dos casos apresentados com o objetivo de devolver a melhor solução;
- Armazenamento dos casos e das respetivas soluções.

Os resultados foram recolhidos após cinco corridas para cada instância apresentada na Tabela 42, e os valores médios normalizados, como referido anteriormente.

Tabela 42 - Resultados obtidos pelo módulo Raciocínio Baseado em Casos

Prob.	*n*	*m*	C_{max} Ótimo	CBR
				C_{max}
La01	10	5	666	0,193
La02	10	5	655	0,239
La05	10	5	593	0,124
La06	15	5	926	0,101
La07	15	5	890	0,156
La08	15	5	863	0,162
La09	15	5	951	0,049
La10	15	5	958	0,028
La11	20	5	1222	0,074
La12	20	5	1039	0,139
La13	20	5	1150	0,052
La14	20	5	1292	0,000
La15	20	5	1207	0,210
La16	10	10	945	0,249
La17	10	10	784	0,244
La18	10	10	848	0,209
La19	10	10	842	0,257
La23	15	10	1032	0,303
La24	15	10	935	0,260
La31	30	10	1784	0,174
La32	30	10	1850	0,203
La33	30	10	1719	0,187
La34	30	10	1721	0,203
ABZ5	10	10	1234	0,204
ABZ6	10	10	943	0,201
ORB4	10	10	1005	0,261
SWV16	50	10	2924	0,105
SWV17	50	10	2794	0,143
SWV18	50	10	2852	0,064
SWV20	50	10	2823	0,064

Comparando os resultados obtidos pela Híper-heurística (*Q-Learning*) com os resultados do módulo de Raciocínio Baseado em Casos, é possível verificar quanto à sua vantagem (Figura 17). Assim, foi utilizado o teste *t* de Student para analisar a significância estatística.

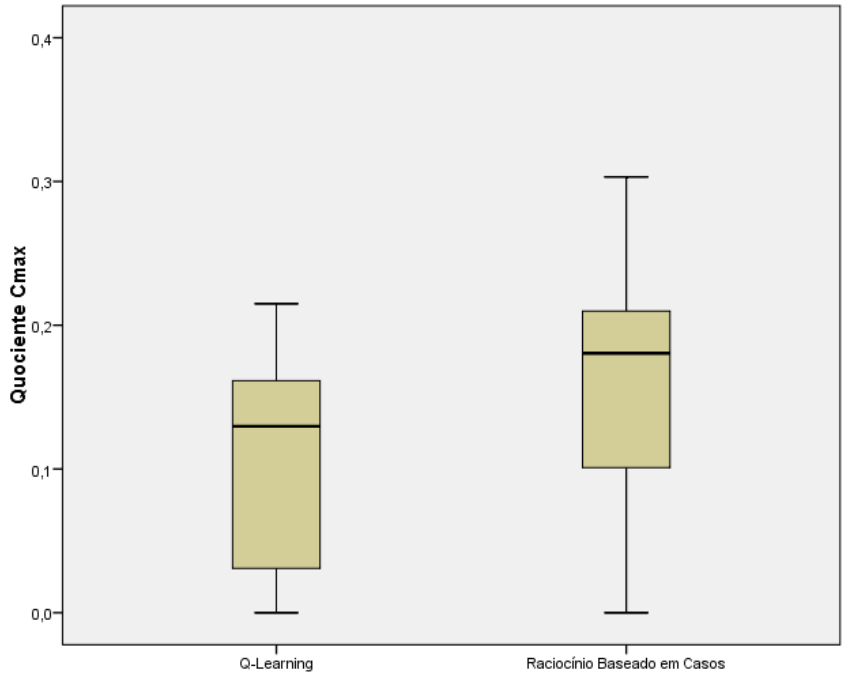

Figura 17 - Comparação do quociente dos valores médios da execução do sistema, entre os resultados obtidos pelo Raciocínio Baseado em Casos e os resultados obtidos pela Híper-heurística

É possível verificar a vantagem da Híper-heurística, através da observação dos valores $t(29) = -7{,}479; p < 0{,}05$ na Tabela 43, pode-se afirmar, com um grau de confiança de 95%, que existem diferenças estatisticamente significativas entres os resultados obtidos pelo Raciocínio Baseado em Casos e os resultados obtidos pela Híper-heurística (critério de paragem de 30 corridas), permitindo concluir sobre a vantagem da Híper-heurística.

Tabela 43 - Resultado do teste t de Student para amostras emparelhadas: Híper-heurística *vs.* Raciocínio Baseado em Casos

	Média da diferença	Desvio padrão da diferença	*t*	Graus de liberdade	*p_value*
Híper-heurística *vs.* CBR	-0,05758	0,04217	-7,479	29	0,000

Também nesta comparação a Híper-heurística apresentou vantagens, permitindo a obtenção de melhores resultados médios, e assim concluir quanto à existência de vantagem estatisticamente significativa da abordagem baseada na Híper-heurística no desempenho do sistema AutoDynAgents. No entanto, é de referir que as técnicas operam de forma diferente, nomeadamente no facto de o CBR utilizar o conceito de memória, ou seja, os casos resolvidos são utilizados na resolução de novos problemas. Mais, o CBR não utiliza um critério de paragem, realizando um ciclo de aprendizagem para cada execução. Contrariamente, o *Q-Learning,* apesar de utilizar memória, esta é reiniciada cada vez que o sistema é executado, precisando de alguns ciclos de aprendizagem para evoluir, sendo esta a possível razão para o desempenho

obtido ter sido melhor para um critério de paragem de 30 corridas. No entanto, seria necessário um futuro estudo para validar a hipótese.

6.3 Sumário do Capítulo

Neste capítulo, foi apresentado o estudo computacional realizado para validação e análise do desempenho do sistema AutoDynAgents, na minimização do C_{max}, com a Híper-heurística proposta e desenvolvida no âmbito desta dissertação de mestrado.

O estudo computacional foi divido em duas partes, na primeira foi efetuado um estudo para validar o desempenho do sistema AutoDynAgents com a Híper-heurística para um determinado critério de paragem, nomeadamente para 10, 20 e 30 corridas. Assim, realizou-se um estudo comparativo entre as corridas para validar qual dos cenários permitiu ao sistema obter um melhor desempenho. O cenário com o melhor desempenho foi utilizado no restante estudo computacional como base de comparação do desempenho da Híper-heurística.

A segunda parte do estudo computacional teve como objetivo validar o desempenho do sistema e eventual vantagem com a incorporação da Híper-heurística. Assim, realizaram-se dois estudos comparativos, o primeiro comparou os resultados obtidos pelo sistema com a Híper-heurística incorporada, face aos resultados obtidos previamente, sem incorporação do mecanismo de aprendizagem. O segundo comparou os resultados obtidos pela Híper-heurística incorporada no sistema com os resultados obtidos pela técnica de aprendizagem Raciocínio Baseado em Casos. A significância estatística dos resultados foi validada para todos os resultados em análise.

Neste contexto, foi possível concluir quanto à existência de vantagens estatisticamente significativas na utilização da Híper-heurística, tendo permitido melhorar o desempenho do sistema AutoDynAgents.

Capítulo 7

Conclusões

Os métodos para a resolução de problemas de Otimização Combinatória, mais especificamente o problema de escalonamento, oferecem muitas oportunidades para a utilização de técnicas de aprendizagem. Em primeiro lugar, porque os dados mudam continuamente, exigindo que os métodos sejam adaptativos. Em segundo lugar, porque os métodos de pesquisa heurística tomam por vezes decisões aleatórias, por exemplo, a seleção da próxima heurística a utilizar. Em terceiro lugar, os dados são muitas vezes baseados em problemas académicos (*benchmarks*), enquanto em problemas do mundo real os dados podem ser significativamente diferentes. No sentido de uma maior aplicabilidade dos métodos de pesquisa heurística, é necessário que estes sejam desenvolvidos de uma forma mais genérica. A presente tese aborda estas questões, onde com a aplicação de técnicas de aprendizagem automática, procura-se desenvolver métodos heurísticos inteligentes e mais adaptativos. Neste capítulo, resumimos o trabalho realizado, apresentamos as limitações e trabalho futuro e por fim as considerações finais.

7.1 Resumo

Nesta dissertação, começou-se por definir o problema de Escalonamento e respetivas abordagens de resolução: métodos de aproximação construtivos (onde se inclui as Regras de Prioridade), por melhoramento, e compostos (Meta-heurísticas e Híper-heurísticas) (Pereira, 2009).

Seguidamente, foi abordada a problemática da Aprendizagem Automática, nas suas diferentes vertentes (a Aprendizagem Supervisionada, a Aprendizagem Não-Supervisionada, e a Aprendizagem por Reforço). Neste contexto, foi realizada a descrição de algumas técnicas de Aprendizagem Automática, nomeadamente as Árvores de Decisão, a Aprendizagem Baseada em Instâncias, a Aprendizagem de Regras, as Redes Neuronais Artificiais, as Redes Bayesianas, os *Support Vector Machines*, o Agrupamento de Dados e o *Q-Learning*.

Em relação ao tema principal desta tese, foi apresentada uma visão geral da metodologia de pesquisa, denominada de Híper-heurística, onde um dos principais objetivos consiste em aumentar o nível de generalidade em que os sistemas de otimização podem operar. A Híper-heurística consiste numa abordagem de alto nível, onde dado um conjunto de Meta-heurísticas, procura selecionar a Meta-heurísticas mais adequada para solucionar um determinado problema. Neste contexto, foi realizada uma descrição mais detalhada sobre as Meta-heurísticas utilizadas, nomeadamente os Algoritmos Genéticos, a Colónia Artificial de Abelhas, a Otimização por Colónia de Formigas, o *Particle Swarm Optimization*, a Pesquisa Tabu e o *Simulated Annealing*.

Foi também apresentada a metodologia de seleção utilizada pela Híper-heurística, assim como a estratégia de aceitação das soluções obtidas.

A arquitetura e o modelo de agentes do sistema Multiagente definido no âmbito do sistema AutoDynAgents foi descrito. O sistema AutoDynAgents consiste num sistema multiagente para a resolução de problemas de escalonamento sujeitos a perturbações. Foi feita uma descrição dos agentes no qual assenta a arquitetura do sistema, bem como de alguns módulos, nomeadamente, o módulo de escalonamento e o mecanismo de coordenação.

Integrada no sistema AutoDynAgents, a Híper-heurística desenvolvida tem como objetivo dotar o sistema com a capacidade de aprendizagem por experiência. Assim, para resolução do problema foi proposta uma Híper-heurística que incorpora um método de Aprendizagem por Reforço, o algoritmo *Q-Learning*. A Híper-heurística é capaz de selecionar e configurar os parâmetros das diferentes Meta-heurísticas, para os diferentes problemas que vão surgindo no sistema. Para um melhor entendimento do funcionamento do módulo, foi apresentado um exemplo ilustrativo.

Finalmente, procedeu-se à realização do estudo computacional para avaliar a influência da Híper-Heurística no desempenho do sistema de escalonamento AutoDynAgents. Este estudo, foi divido em duas partes: na primeira foi efetuado um estudo para validar o desempenho do sistema AutoDynAgents com a Híper-heurística para um determinado critério de paragem, para 10, 20 e 30 corridas. Assim, realizou-se um estudo comparativo para validar qual dos cenários permitiu ao sistema obter um melhor desempenho. Na segunda parte do estudo validou-se o desempenho do sistema com a incorporação da Híper-heurística. Os resultados obtidos pelo sistema com a Híper-heurística incorporada foram comparados com os resultados obtidos previamente, sem incorporação do mecanismo de aprendizagem. Como conclusão genérica, dos resultados obtidos é possível concluir existir vantagem significativa no desempenho do sistema quando introduzida a Hiper-heurística baseda em Q-Learning. Validaram-se ainda os resultados obtidos pela Híper-heurística incorporada no sistema com os resultados obtidos pela técnica de aprendizagem Raciocínio Baseado em Casos. O estudo permitiu verificar a consistência da Híper-heurística.

7.2 Contribuições

Nesta dissertação pretendeu-se dar uma contribuição para a resolução do problema de seleção de Meta-Heurísticas e respetiva parametrização tirando partido das potencialidades das Híper-heurísticas e da Aprendizagem Automática. Neste sentido, foi descrita a especificação de uma Híper-heurística para a seleção de técnicas baseadas na natureza, na resolução do problema de escalonamento de tarefas em sistemas de fabrico, com base em aprendizagem por experiência. O módulo de Híper-heurística desenvolvido utiliza um algoritmo de Aprendizagem por Reforço (*Q-Learning*), que permite dotar o sistema da capacidade de seleção automática da Meta-heurística a usar no processo de otimização, assim como da respetiva parametrização.

Este trabalho surgiu da necessidade do desenvolvimento e especificação de um módulo de aprendizagem que suportasse o sistema AutoDynAgents (POCTI/EMEGIN/66848/2006) no comportamento de auto-parametrização no âmbito do processo de autogestão.

Foi descrita a proposta e o desenvolvimento de uma Híper-heurística para integração com o sistema AutoDynAgents para a resolução do problema de escalonamento. O módulo Híper-heurística foi desenvolvido com base numa técnica de Aprendizagem por Reforço, o algoritmo *Q-Learning*. Este módulo permitiu dotar o sistema com a capacidade de selecionar e definir a parametrização de uma Meta-heurística, na resolução de problemas de escalonamento.

Da análise realizada, aos resultados obtidos no estudo computacional do sistema AutoDynAgents com a Híper-heurística incorporada, comparando-se os resultados obtidos com resultados anteriores e com resultados obtidos por outra técnica de aprendizagem, o Raciocínio Baseado em Casos, foi possível concluir que a utilização deste módulo constitui uma vantagem competitiva. Também, foi possível verificar que as soluções vão melhorando ao longo do tempo, pois a Híper-Heurística vai ficando cada vez mais precisa.

7.3 Limitações e Trabalho Futuro

Como referido por Pereira (2009, 2014), umas das limitações prende-se com o facto de apenas ter sido considerada a heurística *SeqNivel* para a geração da solução inicial das Meta-heurísticas nos casos de inicialização da Base de Dados. A escolha desta heurística deveu-se ao facto da mesma ter obtido anteriormente bons resultados. No entanto, é necessário analisar o comportamento e evolução do sistema com outras heurísticas de geração da solução inicial, de modo a ser possível verificar se a *SeqNivel* é ou não a mais adequada, dependendo da evolução da parametrização das Meta-heurísticas na resolução de problemas de escalonamento diferentes.

Outra limitação refere-se ao facto da Aprendizagem por Reforço não generalizar, ou seja, normalmente um aprendiz deve ser capaz de fazer afirmações gerais sobre todo o ambiente, com base em apenas algumas amostras do meio ambiente. Esse é o objetivo de um algoritmo de aprendizagem. No entanto, a Aprendizagem por Reforço aprende uma ação para cada ponto no espaço (cada estado). Para o *Q-Learning* este aspeto é exponencial, pois este desenvolve uma noção de utilidade para todas as combinações possíveis de estado e ação. Por exemplo, para um problema onde existem 100 estados e seis ações, a tabela *Q*, que relaciona o par estado-ação, vai ter 600 entradas.

Para além disto, o algoritmo *Q-Learning* implementado não utiliza o conceito de memória, ou seja, cada vez que o sistema é executado para resolver um determinado problema a tabela *Q*, é inicializada a zero. O que implica que o processo de aprendizagem começa sempre do início, sem conhecimento anterior. Assim, o sistema precisa de tempo para convergir para uma solução ótima, e isso é visível nos resultados obtidos, pois foi o terceiro cenário, com 30 corridas, que obteve os melhores resultados.

De modo a superar estas limitações, sugere-se para trabalho futuro, a implementação de uma outra técnica da aprendizagem, como redes neuronais, para resolver o problema de generalização, de forma a simplificar o espaço de estados pela classificação dos estados que têm a mesma ação associada. Por fim, propõem-se armazenar a tabela *Q* na base de dados de forma a ser possível manter a memória do processo de aprendizagem e assim diminuir o número de iterações necessárias para alcançar o ótimo.

7.4 Considerações finais

Como considerações finais gostaria apenas de referir que a realização deste trabalho foi muito gratificante, por todo o conhecimento que permitiu adquirir, através do estudo do estado da arte sobre Aprendizagem, e através da implementação da Híper-heurística, que se considera ser um contributo para a melhoria de soluções do sistema AutoDynAgents, bem como um contributo para a evolução do problema da auto-parametrização de Meta-heurísticas, noutros problemas e sistemas.

Bibliografia

ADAMS, J., BALAS, E. & ZAWACK, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science,* 34**,** pp. 391–401.

ALONSO, E., INVERNO, M. D., KUDENKO, D., LUCK, M. & NOBLE, J. 2001. Learning in Muli-Agent Systems. *The Knowledge Engineering Review, Cambridge University Press,* 16(3)**,** pp. 277-284.

ALPAYDIN, E. 2004. *Introduction to machine learning*, The MIT Press**,** pp. 584.

APPLEGATE, D. & COOK, W. 1991. A computational study of the job-shop scheduling instance. *ORSA Journal on Computing,* 3**,** pp. 149–156.

BAI, R., BLAZEWICZ, J., BURKE, E. K., KENDALL, G. & MCCOLLUM, B. 2007. A simulated annealing hyper-heuristic methodology for fexible decision support. *Tech. Rep. NOTTCS-TR-2007-8, School of CSiT, University of Nottingham*.

BAKER, K. 1974. Introduction to sequencing and scheduling. *Business & Economics . Wiley***,** pp. 305

BATTITI, R. & BRUNATO, M. 2007. Reactive search: Machine learning for memory-based heuristics. *Gonzalez TF (ed) Approximation Algorithms and Metaheuristics. Taylor and Francis Books (CRC Press)* 21**,** pp. 1-17.

BLUM, C. & ROLI, A. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys,* 35**,** pp. 268-308.

BOX, J. F. 1987. Guinness, Gosset, Fisher, and Small Samples. *Statistical Science,* 2**,** pp. 45-5.

BRANKE, J. 2000. Efficient evolutionary algorithms for searching robust solutions. *In Proceedings of the Fourth International Conference on Adaptive Computing in Design and Manufacture. Plymouth, Springer*.

BRUCKER, P. 2001. *Scheduling Algorithms*, Springer**,** pp. 371.

BURKE, E., KENDALL, G. & SOUBEIGA, E. 2003a. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics* 9**,** pp. 451-470.

BURKE, E., MEISELS, A., PETROVIC, S. & QU, R. 2002. Case based heuristic selection for examination timetabling. *In: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL 2002)***,** pp. 277–281.

BURKE, E. K., HART, E., KENDALL, G., NEWALL, J., ROSS, P. & SCHULENBURG, S. 2003b. Hyper-heuristics: An emerging direction in modern search technology. *Handbook of*

Metaheuristics (F. Glover and G. Kochenberger, eds.), Kluwer Academic Publishers, pp. 457–474.

BURKE, E. K., HYDE, M., KENDALL, G., OCHOA, G., OZCAN, E. & QU, R. 2013. Hyper-heuristics: A Survey of the State of the Art. *Journal of the Operational Research Societ,* 64, pp. 1695-1724.

BURKE, E. K., KENDALL, G., OCHOA, G., MISIR, M., OZCAN, E. & WOODWARD, J. 2009. Handbook of meta-heuristics, international series in operations research & management science. 146, pp. 449–468.

BURKE, E. K., SILVA, J. D. L. & SOUBEIGA, E. 2005. Multi-objective hyper-heuristic approaches for space allocation and timetabling. *In: T. Ibaraki, K. Nonobe, M. Yagiura (eds.) Metaheuristics: Progress as Real Problem Solvers. Selected Papers from the 5th Metaheuristics International Conference (MIC 2003). Operations Research/Computer Science Interfaces Series,* 32, pp. 129-158.

CARBONELL, J. G., MICHALSKI, R. S. & MITCHELL, T. M. 1983. *Machine Learning: An Artificial Intelligence Approach*, M. Kaufmann, pp. 572

CERNY, V. 1985. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications,* 45, pp. 41-51.

CHAKHLEVITCH, K. & COWLING, P. 2008. Hyperheuristics: Recent developments. *In: Cotta C. Sevaux M. Sorensen K (eds) Adaptive and Multilevel Metaheuristics,* 136, pp. 3–29.

CHONG, C. S. & LOW, M. Y. H. 2006. A bee colony optimization algorithm to job shop scheduling. *Proceedings of the 2006 Winter Simulation Conference*, pp. 1954–1961.

CHOU, L.-D., LIU, T.-C., LI, D. C., CHEN, Y.-S., LEONG, M. T., LEE, P.-H. & LIN, Y.-C. 2011. Development of a Game-based Learning System Using Toy Robots. *Advanced Learning Technologies (ICALT), 2011 11th IEEE International Conference*, pp. 202-204.

CONOVER, W. J. 1999. *Practical nonparametric statistics: John Wiley&Sons*, Wiley, pp. 592.

CONWAY, W. R., MAXWELL, W. L. & MILLER, L. W. 1967. Theory of scheduling. *Addison-Wesley Publishing Company*, pp. 294

CORTES, C. & VAPNIK, V. 1995. Support-vector networks. *Machine Learning,* 30(3), pp. 273.

COVER, T. M. & HART, P. E. 1967. Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, pp. 13, 21-27.

COWLING, P., KENDALL, G. & HAN, L. 2002. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. *In Proceedings of 2002 Congress on Evolutionary Computation (CEC 2002). IEEE Computer Society Press*, pp. 1185–1190.

COWLING, P., KENDALL, G. & SOUBEIGA, E. 2000. Hyperheuristic Approach to Scheduling a Sales Summit. *In Selected papers of Proceedings of the Third International Conference of Practice and Theory of Automated Timetabling (PATAT2000),* 2079**,** pp. 176-190.

CROWSTON, W. B., GLOVER, F., THOMPSON, G. L. & TRAWICK, J. D. 1963. Probabilistic and parametric learning combinations of local job shop scheduling rules. *Carnegie-Mellon University,* (117).

DASARATHY, B. V. 1980. Nosing around the neighborhood: A new system structure and classification rule for recognition in partially exposed environments. *Pattern Analysis and Machine Intelligence***,** pp. 2, 67-71.

DELL'AMICO, M. & TRUBIAN, M. 1993. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research,* 41(1-4)**,** pp. 231–252.

DORIGO, M., MANIEZZO, V. & COLORNI, A. 1991. The ant system: an autocatalytic optimizing. *Technical Report TR91-016. Politecnico di Milano***,** pp. 1-21.

DORIGO, M. & STÜTZLE, T. 2004. *Ant Colony Optimization*, The MIT Press.

DOWSLAND, K. A., SOUBEIGA, E. & BURKE, E. K. 2007. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research,* 179(3)**,** pp. 759-774.

DUARTE, J. 2008. Agrupamento de Dados com Restrições. *Tese de Mestrado, Instituto Superior de Engenharia do Porto, Portugal*.

FANG, H., ROSS, P. & CORNE, D. 1993. A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems. *In: Proceedings of the Fifth International Conference on Genetic Algorithms, S. Forrest (ed.), San Mateo: Morgan Kaufmann***,** pp. 375–382.

FAWCETT, T. & PROVOST, F. 1996. Combining Data Mining and Machine Learning for Effective User Profiling *KDD-96 Proceedings***,** pp. 8-13.

FERA, M., FRUGGIERO, F., LAMBIASE, A., MARTINO, G. & NENNI, M. E. 2013. Production Scheduling Approaches for Operations Management. *Operations Management, Prof. Massimiliano Schiraldi (Ed.)*.

FISHER, H. & THOMPSON, G. L. 1961. Probabilistic learning combinations of local job-shop scheduling rules. *In: Factory Scheduling Conference, Carnegie Institue of Technology. Carnegie Institute of Technology*.

FISHER, H. & THOMPSON, G. L. 1963. Probabilistic learning combina-tions of local job-shop scheduling rules. *In: J. F. Muth and G. L. Thompson (eds) Industrial Scheduling. Prentice Hall, Englewood Cliffs***,** pp. 225–251.

GATES, G. W. 1972. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, pp. 431-433.

GHAHRAMANI, Z. 2008. Unsupervised learning algorithms are designed to extract structure from data. *IOS Press,* 176, pp. 1-8.

GLOVER, F. 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research,* 13(5), pp. 533–549.

GLOVER, F. & KOCHENBERGER, G. A. 2003. Handbook of metaheuristics. *Springer*, pp. 2, 12.

GLOVER, F. & LAGUNA, M. 1997. *Tabu search*, Kluwer Academic Publishers Norwell.

GOOD, I. J. 1950. *Probability and the Weighing of Evidence,* London, Charles Griffin and Company, pp. 119.

GRATCH, J. & CHIEN, S. 1996. Adaptive problem-solving for large-scale scheduling problems: A case study. *Journal of Artificial Intelligence Research,* 4, pp. 365–396.

GRATCH, J., CHIEN, S. & DEJONG, G. 1993. Learning search control knowledge for deep space network scheduling. *In the Proceedings of the Tenth International Conference on Machine Learning*, pp. 135–142.

GUNN, S. 1998. Support Vector Machines for Classification and Regression. *ISIS Technical Report*.

HAIPENG, Z., MITSUO, G., SHIGERU, F. & WOO, K. K. 2004. Hybrid Ant Colony Optimization for Job-shop Scheduling Problem. *Faji Shisutemu Shinpojiumu Koen Ronbunshu,* 20, pp. 304-05.

HART, E. & ROSS, P. 1998. A heuristic combination method for solving job-shop scheduling problems., pp. 845–854.

HART, E., ROSS, P. & NELSON, J. A. D. 1998. Solving a real-world problem using an evolving heuristically driven schedule builder., pp. 61–80.

HART, P. E. 1968. The condensed nearest neighbor rule. *Institute of Electrical and Electronics Engineers and Transactions on Information Theory*, pp. 14, 515-516.

HOLLAND, J. H. 1975. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. *University of Michigan Press*, pp. 183.

HUTTER, F., HOOS, H. H. & STUTZLE, T. 2007. Automatic algorithm con¯guration based on local search. *AAAI, AAAI Press,*, pp. 1152-1157.

JAIN, A. K. 2010. Data Clustering: 50 Years Beyond K-Means. *Pattern Recognition Letters,* 31(8), pp. 651-666.

KARABOGA, D. 2005. An Idea Based On Honey Bee Swarm for Numerical Optimization. *Technical Report-TR06,Erciyes University, Engineering Faculty, Computer Engineering Department*.

KARABOGA, D. & AKAY, B. 2011. A modified artificial bee colony (abc) algorithm for constrained optimization problems. *Applied Soft Computing. Elsevier,* 11(3)**,** pp. 3021–3031.

KARABOGA, D. & BASTURK, B. 2008. On The performance of Artificial Bee Colony (ABC) algorithm. *Applied Soft Computing,* 8(1)**,** pp. 687–697.

KENNEDY, J. & EBERHART, R. 1995. Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks,* 4**,** pp. 1942–1948.

KEPHART, J. & CHESS, D. 2003. The Vision of Autonomic Computing. *Computer Magazine,* 36(1)**,** pp. 41-50.

KIRKPATRICK, S., GELATT, C. D. & VECCHI, M. P. 1983. Optimization by Simulated Annealing. *Science,* 220**,** pp. 671–680.

KOTSIANTIS, S. B. 2007. Supervised Machine Learning: A Review of Classification Techniques. *Informatica,* 31**,** pp. 249-268.

KRISHNA, K., GANESHAN, K. & RAM, D. J. 1995. Distributed simulated annealing algorithms for job shop scheduling. *Systems, Man and Cybernetics, IEEE Transactions,* 25(7)**,** pp. 1102-09.

LADDAGA, R. 2002. Creating robust software through self-adaptation. *Intelligent Systems and their Applications. IEEE* 14(3)**,** pp. 26 - 29.

LAWRENCE, S. 1984. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. *Pennsylvania: Graduate School of Industrial Administration. Carnegie-Mellon University*.

LEE, C. Y., PIRAMUTHU, S. & TSAI, Y. K. 1997. Job shop scheduling with a genetic algorithm and machine learning. *International Journal of Production Research,* 35**,** pp. 1171–1191.

LIU, H., ABRAHAM, A., CHOI, O. & MOON, S. H. 2006. Variable Neighborhood Particle Swarm Optimization for Multi-objective Flexible Job-Shop Scheduling Problems. *In 6th international conference, SEAL 2006,* 4247**,** pp. 197-204.

LOBO, F. G., LIMA, C. F. & MICHALEWICZ, Z. 2007. Parameter Setting in Evolutionary Algorithms. *Studies in Computational Intelligence, Springer,* 54.

MADUREIRA, A. 2009. Técnicas Emergentes de Optimização no Suporte à Tomada de Decisão. *Documento apresentado no concurso de provas públicas para Professor Coordenador. ISEP/IPP*.

MADUREIRA, A., FALCAO, D. & PEREIRA, I. 2012. Ant Colony System based approach to single machine scheduling problems: Weighted tardiness scheduling problem. *2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC)***,** pp. 86-91.

MADUREIRA, A. & PEREIRA, I. 2010. Self-Optimization for Dynamic Scheduling in Manufacturing Systems. *Technological Developments in Networking, Education and Automation. Springer***,** pp. 421–426.

MADUREIRA, A. & PEREIRA, I. 2011. Intelligent Bio-Inspired System for Manufacturing Scheduling under Uncertainties. *International Journal of Computer Information Systems and Industrial Management Applications,* 3**,** pp. 72-79.

MADUREIRA, A., PEREIRA, I. & FALCÃO, D. 2013. Cooperative Scheduling System with Emergent Swarm Based Behavior. *Álvaro Rocha et al. (eds.): Advances in Information Systems and Technologies, Advances in Intelligent Systems and Computing, Springer,* 206**,** pp. 661-671.

MADUREIRA, A., PEREIRA, I., PEREIRA, P. & ABRAHAM, A. 2014. Negotiation Mechanism for Self-Organized Scheduling System with Collective Intelligence. *Neurocomputing, Elsevier,* 132**,** pp. 97–110.

MADUREIRA, A., PEREIRA, I., SOUSA, N., ÁVILA, P. & BASTO, J. 2011a. Scheduling a Cutting and Treatment Stainless Steel Sheet Line with Self-Management Capabilities. *Computational Intelligence for Engineering Systems: Emergent Applications. Springer,* 46**,** pp. 34-47.

MADUREIRA, A., SANTOS, J. & PEREIRA, I. 2007. MASDSheGATS: A Prototype System for Dynamic Scheduling. *In 6th WSEAS Int.Conf. on COMPUTATIONAL INTELLIGENCE, MAN-MACHINE SYSTEMS and CYBERNETICS (CIMMACS '07)*.

MADUREIRA, A., SANTOS, J. & PEREIRA, I. 2008. Inter-Machine Cooperation Mechanism for Dynamic Scheduling. *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE'2008)***,** pp. 483-488.

MADUREIRA, A., SANTOS, J. & PEREIRA, I. 2009. MASDSheGATS – Scheduling System for Dynamic Manufacturing Environments. *In S. Ahmed & M.N. Karsity, eds. MultiAgent Systems. In-Tech*.

MADUREIRA, A., SOUSA, N. & PEREIRA, I. 2011b. Self-organization for Scheduling in Agile Manufacturing. *10th IEEE International Conference on Cybernetic Intelligent Systems 2011 (IEEE CIS 2011)***,** pp. 38-43.

MADUREIRA, A. M. 2003. *Aplicação de Meta-Heurísticas ao Problema de Escalonamento em Ambiente Dinâmico de Produção Discreta* Tese de Doutoramento, Universidade do Minho, Braga, Portugal.

MATURANA, J., FIALHO, A., SAUBION, F., SCHOENAUER, M. & SEBAG, M. 2009. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. *Proc. IEEE Congress on Evolutionary Computation CEC '09***,** pp. 365-372.

MCCULLOCH, W. & PITTS, W. 1943. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics,* 5(4)**,** pp. 115–133.

MEHTA, S. V. & UZSOY, R. 1999. Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing,* 12(1)**,** pp. 15-38.

METROPOLIS, N., ROSENBLUTH, A. W., MARSHALL, N., TELLER, A. H. & TELLER, E. 1953. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics,* 21(6)**,** pp. 1087.

MINSKY, M. & PAPERT, S. 1969. Perceptrons: An Introduction to Computational Geometry. pp. 258

MITCHELL, T. M. 1997. Machine Learning.

MITCHELL, T. M. 2006. The Discipline of Machine Learning. *Technical report CMU-ML-06-108, Carnegie Mellon University***,** pp. 1-7.

MLADENOVIC, N. & HANSEN, P. 1997. Variable neighborhood search. *Computers and Operations Research,* 24(11)**,** pp. 1097-1100.

MORO, S., LAUREANO, R. & CORTEZ, P. 2011. Using data mining for bank direct marketing: An application of the crisp-dm methodology. *In P. Novais et al. (Eds.), Proceedings of the European Simulation and Modelling Conference - ESM'2011***,** pp. 117-121.

MORTON, E. & PENTICO, D. W. 1993. *Heuristic Scheduling Systems*, John Wiley & Sons**,** pp. 720

NALLUR, V. & BAHSOON, R. 2013. A decentralized self-adaptation mechanism for service-based applications in the cloud. *Software Engineering, IEEE* 39(5)**,** pp. 591-612

NAREYEK, A. 2003. Choosing search heuristics by non-stationary reinforcement learning. *In: Resende, M., de Sousa, J. (eds.) Metaheuristics: Computer decision-making. Kluwer Academic Publishers***,** pp. 523–544.

NEAPOLITAN, R. E. 1989. *Probabilistic reasoning in expert systems: theory and algorithms*, Wiley**,** pp. 433

NIESE, R., AL-HAMADI, A., HEUER, M., B. MICHAELIS & MATUSZEWSKI, B. 2011. Machine vision based recognition of emotions using the circumplex model of affect. *Multimedia Technology (ICMT), 2011 International Conference***,** pp. 6424-6427.

NORENKOV, I. & GOODMAN, E. 1997. Solving scheduling problems via evolutionary methods for rule sequence optimization. *Soft Computing in Engineering Design and Manufacturing***,** pp. 350-355.

OSMAN, I. & KELLY, J. 1996. *Meta-Heuristics: Theory and Applications*, Springer**,** pp. 690.

OUELHADJ, D. & PETROVIC, S. 2008. A Survey of Dynamic Scheduling in Manufacturing Systems. *Journal of Scheduling,* 12(4)**,** pp. 417-431.

OUELHADJ, D. & PETROVIC, S. 2009. Asynchronous cooperative hyper-heuristic search. *Proceedings of the International Conference on Arti¯cial Intelligence,***,** pp. 78-84.

OUELHADJ, D. & PETROVIC, S. 2010. A cooperative hyper-heuristic search framework. *Journal of Heuristics***,** pp. 78-84.

OZCAN, E., BILGIN, B. & KORKMAZ, E. E. 2008. A comprehensive survey of hyperheuristics. Intelligent Data Analysis. *Intelligent Data Analysis,* 12(1)**,** pp. 3-23

PANAIT, L. & LUKE, S. 2005. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems,* 11(3)**,** pp. 387-434.

PARASHAR, M. & HARIRI, S. 2006. Autonomic Computing: Concepts, Infrastructure, and Applications. *CRC Press*.

PEARL, J. 1985. Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning. *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, CA***,** pp. 329–334.

PEARL, J. 1988. *Probabilistic Reasoning in Intelligent Systems,* San Francisco, CA, Morgan Kaufmann Publishers**,** pp. 552

PEREIRA, I. 2009. *Aspectos de aprendizagem em optimização.* Tese de Mestrado, Instituto Superior de Engenharia do Porto, Portugal.

PEREIRA, I. 2014. *Sistema Inteligente para Escalonamento Assistido por Aprendizagem.* Tese de Doutorament, UTAD, Vila Real.

PEREIRA, I. & MADUREIRA, A. 2013. Self-Optimization module for Scheduling using Case-based Reasoning. *Applied Soft Computing***,** pp. 1419–1432.

PHAM, D. T. & KARABOGA, D. 2000. *Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks*, Springer**,** pp. 302

PHAM, D. T., KOÇ, E., LEE, J. Y. & PHRUEKSANANT, J. 2007. Using the bees algorithm to schedule jobs for a machine. *Proceedings of eighth international conference on laser metrology, CMMandmachine tool performance***,** pp. 430–439.

PIRLOT, M. 1996. General Local Search Method. *European Journal of Operational Research,* 92**,** pp. 493-522.

PISINGER, D. & ROPKE, S. 2007. A general heuristic for vehicle routing problems. *Computers and Operations Research,* 34**,** pp. 2403-2435.

PONGCHAIRERKS, P. & KACHITVICHYANUKUL, V. 2009. A Particle Swarm Optimization Algorithm on Job- Shop Scheduling Problems with Multi-Purpose Machines. *Asia-Pacific Journal of Operational Research (APJOR),* 26(2)**,** pp. 161-84.

PONNAMBALAM, G., JAWAHAR, N. & ARAVINDAN, P. 1999. A simulated annealing algorithm for job shop scheduling. *Production Planning and Control,* 10(8)**,** pp. 767-77.

PONNAMBALAM, S. G., ARAVINDAN, P. & RAJESH, S. V. 2000. A tabu search algorithm for job shop scheduling. *The International Journal of Advanced Manufacturing Technology,* 6(10)**,** pp. 765–771.

REEVES, C. R. 1995. *Modern heuristic techniques for combinatorial problems*, John Wiley & Sons**,** pp. 320.

RICE, J. R. 1976. The algorithm selection problem. *Advances in Comput,* 15**,** pp. 65-118.

ROSENBLATT, F. 1958. The perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review,* 65(6)**,** pp. 386-408.

ROSS, P. 2005. Hyper-heuristics. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques (E. K. Burke and G. Kendall, eds.)***,** pp. 529-556.

ROSS, P., SCHULENBURG, S., ÁZQUEZ, J. G. M.-B. & HART, E. 2002. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. *In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)***,** pp. 942–948.

RUMELHART, D. E., HINTON, G. E. & WILLIAMS, R. J. 1986. *Learning Internal Representations by Error Propagation,* Cambridge, MA, USA, MIT Press**,** pp. 318-362

SHA, D. Y. & HSU, C. 2006. A hybrid particle swarm optimization for job shop scheduling problem. *Comput. Ind. Eng.,* 51(4)**,** pp. 791-808.

SMITH-MILES, K. 2008. Towards insightful algorithm selection for optimisation using meta-learning concepts. *Conference: Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence)***,** pp. 4118-4124.

STEINHOFEL, K., ALBRECHT, A. & WONG, C. K. 1999. Two simulated annealing-based heuristics for the job shop scheduling problem. *European Journal of Operational Research,* 118(3)**,** pp. 524-48.

STONE, P. & VELOSO, M. 2000. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robotics,* 8(3).

STORER, R. H., WU, S. D. & VACCARI, R. 1992a. New search spaces for sequencing instances with application to job shop scheduling. *Management Science,* 38**,** pp. 1495–1509.

STORER, R. H., WU, S. D. & VACCARI, R. 1992b. New search spaces for sequencing problems with application to job shop scheduling. 38**,** pp. 1495–1509.

SUTTON, R. S. & BARTO, A. G. 1998. Reinforcement Learning: An Introduction. MIT Press.

TAILLARD, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research,* (64)**,** pp. 278-285.

TALBI, E. G. 2009. *Metaheuristics: From Design to Implementation*, John Wiley and Sons**,** pp. 593.

TSENG, J. C. R. & HWANG, G.-J. 2007. Development of an automatic customer service system on the internet. *Electronic Commerce Research and Applications,* 6(1)**,** pp. 19-28.

UTGOFF, P. E. & MITCHELL, T. M. 1982. Acquisition of Appropriate Bias for Inductive Concept Learning. *Proceedings of the Second National Conference on Artificial Intelligence***,** pp. 414-417.

VENTRESCA, M. & OMBUKI, B. M. 2004. Ant Colony Optimization for Job Shop Scheduling Problem. *Technical Report # CS-04-04. Brock University*.

WANG, D., BASTANI, F. B., YEN, I.-L. & PAUL, R. A. 2005. An approach for designing highly adaptable process-control systems. *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005, Eighth IEEE International Symposium***,** pp. 106- 113.

WATKINS, C. J. C. H. 1989. Learning from Delayed Rewards. *PhD thesis, Cambridge University of Michigan Press***,** pp. 10, 11, 21.

WATKINS, C. J. C. H. & DAYAN, P. 1992. Q-learning. Machine Learning. 8**,** pp. 279–292.

WERBOS, P. J. 1974. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. *PhD thesis, Harvard University*.

YAMADA, T. & NAKANO, R. 1992. A genetic algorithm applicable to large-scale job-shop instances. *In: R. Manner, B. Manderick (Eds.), Parallel Instance Solving from Nature 2, North-Holland, Amsterdam***,** pp. 281–290.

YAMADA, T. & NAKANO, R. 1994. Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search. *EICE technical report. Artificial intelligence and knowledge-based processing,* 94(374)**,** pp. 77-84.

YEN, G. G. & IVERS, B. 2009. Job shop scheduling optimization through multiple independent particle swarms. *International Journal of Intelligent Computing and Cybernetics,* 2(1)**,** pp. 5-33.

YOSHIKAWA, M. & TERAI, H. 2006. A Hybrid Ant Colony Optimization Technique for Job-Shop Scheduling Problems. *In Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)***,** pp. 95-100.

ZADEH, L. A. 1965. Fuzzy Set. *Info. Control,* 81**,** pp. 53-338.

ZWAAN, S. V. D., PAIS, A. R. & MARQUES, C. 1999. Ant Colony Optimisation for Job Shop Scheduling. *In Proceedings of the'99 Workshop on Genetic Algorithms and Artficial Life GAAL'99***,** pp. 1-8.