Universidade de Trás-os-Montes e Alto Douro

Sistema Inteligente para Escalonamento Assistido por Aprendizagem

Tese de Doutoramento em

Engenharia Eletrotécnica e de Computadores

Ivo André Soares Pereira

Ana Maria Dias Madureira Pereira

José Paulo Barroso de Moura Oliveira

Vila Real, 2014

Universidade de Trás-os-Montes e Alto Douro

Sistema Inteligente para Escalonamento Assistido por Aprendizagem

Tese de Doutoramento em

Engenharia Eletrotécnica e de Computadores

Ivo André Soares Pereira

Ana Maria Dias Madureira Pereira

José Paulo Barroso de Moura Oliveira

Composição do Júri:

Doutor Vitor Manuel de Jesus Filipe, Universidade de Trás-os-Montes e Alto Douro

Doutor Francisco José Baptista Pereira, Instituto Superior de Engenharia de Coimbra

Doutor Paulo Jorge Freitas de Oliveira Novais, Universidade do Minho

Doutor Manuel José Cabral dos Santos Reis, Universidade de Trás-os-Montes e Alto Douro

Doutor Eduardo José Solteiro Pires, Universidade de Trás-os-Montes e Alto Douro

Vila Real, 2014

Este trabalho foi financiado pela FCT - Fundação para a Ciência e a Tecnologia, através do programa de financiamento QREN – POPH - Tipologia 4.1 – Formação Avançada comparticipado pelo Fundo Social Europeu e por fundos do MCTES.

(Bolsa individual com a referência SFRH/BD/63404/2009) entre 2010 e 2014.

Esta tese foi redigida ao abrigo do
Acordo Ortográfico de 1990.

«À minha família e amigos»

Agradecimentos

Gostaria de agradecer a todas as pessoas que, de alguma forma, deram o seu contributo para que realização deste trabalho fosse possível.

Começo por agradecer à minha orientadora, Doutora Ana Madureira, pela sua, compreensão, disponibilidade e preocupação ao longo dos últimos 7 anos de trabalho em conjunto, e também pelos conselhos e críticas construtivas no processo de revisão desta tese.

Ao meu co-orientador, Doutor Paulo de Moura Oliveira, pelo seu interesse e disponibilidade que sempre demonstrou e pelas críticas construtivas, sobretudo no trabalho de revisão deste documento.

Ao GECAD (Grupo de Engenharia de Conhecimento e Apoio à Decisão), pela oportunidade que me concedeu ao realizar este trabalho e por todas as condições disponibilizadas para o efeito.

A todos os meus colegas, sem exceção, pelo apoio, sugestões, e críticas em várias questões relacionadas com este trabalho, mas também pelo companheirismo e amizade que demonstraram ao longo dos últimos quatro anos.

A todos os meus amigos por todo o apoio demonstrado e pela paciência e compreensão nos momentos mais difíceis. À Raquel pela preciosa ajuda em algumas imagens.

À egrégora SwáSthya, principalmente aquela presente na escola Sentidos Urbanos, por toda a força e energia positiva que me proporcionaram.

À minha família, em especial aos meus pais e ao meu irmão, por todo o apoio e compreensão ao longo de toda a minha vida.

A todos, o meu sincero obrigado!

Resumo

Em ambientes reais de produção, existe o problema de atribuição de tarefas a determinadas máquinas de forma eficaz e eficiente, que se designa por Escalonamento. A principal motivação deste trabalho surgiu da necessidade de desenvolvimento de abordagens que sejam capazes de controlar, coordenar e otimizar, de forma adaptativa, a resolução dos diferentes desafios existentes no problema de Escalonamento.

Este trabalho de doutoramento foi inserido no âmbito do projeto de I&D *AutoDynAgents*, que consiste num Sistema Multiagente para a resolução autónoma, distribuída e cooperativa de problemas de escalonamento de tarefas em sistemas de produção. Este sistema incorpora conceitos da Computação Autónoma e Meta-heurísticas para a construção de planos de escalonamento.

A afinação de parâmetros nas Meta-heurísticas pode permitir uma maior flexibilidade e robustez mas requer uma inicialização cuidadosa. Os parâmetros podem ter uma influência significativa na eficiência e eficácia do processo de pesquisa. Não se torna óbvia a definição *a priori* dos valores dos parâmetros, que dependem do problema, das instâncias a tratar e do tempo disponível para a resolução do problema. Além disso, não existem valores "universais" para os parâmetros das Meta-heurísticas. Existe uma opinião generalizada que a sua afinação deve resultar de um cuidadoso esforço experimental.

Surge assim a necessidade de implementação de um módulo para a seleção autónoma de uma determinada Meta-heurística, e respetiva especificação automática dos parâmetros, para a resolução de novos problemas de escalonamento de tarefas em sistemas de produção. Este módulo incorpora técnicas de aprendizagem, de modo a dotar o sistema da capacidade de aprender com a experiência adquirida na resolução de casos anteriores similares: *Racing*, Raciocínio baseado em Casos, e híbrida.

De modo a avaliar a contribuição deste trabalho, foi desenvolvido um estudo computacional das várias abordagens seguidas, com destaque para a abordagem híbrida que agrega *Racing* e Raciocínio baseado em Casos. Foi possível concluir acerca da vantagem estatisticamente significativa na utilização de aprendizagem na autoparametrização de Meta-heurísticas.

Palavras-chave: Escalonamento, Meta-heurísticas, Afinação de parâmetros, Sistemas Multiagente, Computação Autónoma, *Racing*, Raciocínio baseado em Casos

Abstract

In real manufacturing environments, there is the problem of assigning tasks to specific machines, effectively and efficiently, which is known as scheduling. The main motivation of this paper came from the need to develop approaches that are able to adaptively control, coordinate and optimize the resolution of the different current challenges in the problem of scheduling.

This PhD thesis was done as part of the R&D AutoDynAgents project, consisting of a Multi-Agent System for autonomous, distributed and cooperative resolution of task scheduling problems in production systems. This system incorporates concepts of Autonomic Computing and Metaheuristics for building scheduling plans.

The tuning of parameters in Metaheuristics can enable greater flexibility and robustness but requires a careful initialization. The parameters can have a significant influence on the efficiency and effectiveness of the search process. A priori definition of the parameter values does not become obvious, which depends on the problem, instances and time available to solve the problem. Furthermore, there are no “universal” values for the parameters of the Metaheuristics. There is a widespread opinion that its tuning should result from careful experimental effort.

Thus leads to the need of implementing a module for the autonomous selection of Metaheuristics, and automatic specification of the respective parameters, to solve new problems of task scheduling in production systems. This module incorporates learning techniques in order to provide the system the ability to learn from the experience acquired in solving previous similar cases: Racing, Case-based Reasoning, and hybrid.

In order to evaluate the contribution of this work, a computational study of the different followed approaches was developed, with emphasis on the hybrid approach that combines Racing and Case-based Reasoning. It was possible to conclude about the statistically significant advantage in the use of learning in Metaheuristics self tuning.

Keywords: Scheduling, Metaheuristics, Parameter tuning, Multi-Agent Systems, Autonomic Computing, Racing, Case-based Reasoning

Índice geral

Índice de algoritmos

Índice de figuras

Índice de tabelas

Glossário

Agente	Sistema que age de forma autónoma num determinado meio e que procura satisfazer os seus próprios objetivos, através de sensores e atuadores.
Aprendizagem Automática	Área de investigação da Inteligência Artificial que se dedica ao desenvolvimento de técnicas e algoritmos para dotar os sistemas da capacidade de aprendizagem.
Autoparametrização	Definição dos parâmetros das Meta-heurísticas, de forma autónoma.
***Cmax* (*makespan*)**	Tempo de conclusão de um plano de Escalonamento.
Computação Autónoma	Paradigma de computação, proposto pela IBM, em que sistemas informáticos são embutidos com mecanismos de manutenção quotidiana, com o objetivo de automatizar essa manutenção.
Escalonamento	Problema de afetação no tempo de tarefas a determinadas máquinas, sujeitas a algumas restrições.
Meta-heurísticas	Métodos iterativos ou recursivos com o objetivo de obter soluções o mais próximo possível do ótimo global, para um determinado problema de otimização.
Racing	Método de Aprendizagem Automática que avalia os candidatos e liberta aqueles que aparentam ser menos promissores, durante o processo de avaliação.
Raciocínio baseado em Casos	Metodologia de Inteligência Artificial com o objetivo de resolver novos problemas através do uso de informação acerca das soluções de problemas anteriores similares.
SeqNivel	Regra de prioridade onde a sequência de operações é definida pela ordenação não-decrescente dos níveis a que pertencem as operações numa gama operatória.
Sistema Multiagente	Sistema constituído por vários agentes, os quais interagem entre si através da troca de mensagens.

Capítulo 1. Introdução

1.1. Motivação e enquadramento

Os ambientes reais de produção apresentam um nível elevado de incerteza, os processos apresentam requisitos específicos e detalhados, e os objetivos de gestão são diversos, dinâmicos e, por vezes, conflituosos. Neste tipo de ambientes de produção, é importante a capacidade de, eficaz e eficientemente, se poder atribuir tarefas a máquinas, sujeito a restrições. Este problema de Otimização Combinatória designa-se por problema de Escalonamento.

No mercado global atual e altamente competitivo, as empresas devem estar conscientes de oportunidades momentâneas de mercado, e reagir rapidamente e corretamente aos pedidos dos clientes. A capacidade para lidar com um grande número de encomendas de pequenas quantidades aumenta o *stress* do processo de escalonamento. Encontrar boas soluções para os problemas de escalonamento torna-se muito importante para os sistemas reais de produção dado que a taxa e os custos de produção são muito dependentes dos planos de escalonamento usados para controlar o fluxo de trabalho através do sistema.

Os problemas de Otimização Combinatória surgem quando existe a necessidade de se selecionar, de um conjunto de dados discreto e finito, o melhor subconjunto que satisfaz determinados critérios de natureza económica-operacional. O grande desafio destes problemas passa por produzir, em tempo competitivo, soluções o mais próximo possível das soluções ótimas. 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.*

De entre as várias abordagens de resolução de problemas de Otimização Combinatória, onde se inclui o problema de Escalonamento, salientam-se os métodos de aproximação, nos quais o objetivo passa por encontrar soluções satisfatórias em tempos de execução aceitáveis. Incluem-se nesta categoria as Meta-heurísticas, muitas das quais consistem em métodos inspirados na natureza para pesquisa de soluções o mais próximo possível do ótimo global de um determinado problema.

As Meta-Heurísticas têm vindo a ganhar popularidade e são cada vez mais usadas em domínios cuja complexidade e necessidade de tomada de decisão em tempo útil tornou

o uso de técnicas exatas incomportável. Uma das tarefas cruciais para alcançar um bom desempenho no uso de Meta-Heurísticas é a especificação dos parâmetros. Esta tarefa de parametrização é normalmente levada a cabo pelo utilizador, com base em experiência ou através de tentativa-erro guiada por regras de ouro. Esta abordagem, além de ser extremamente entediante, é suscetível ao erro, dificilmente reprodutível, e dispendiosa (Birattari, 2009; Cotta *et al.*, 2008).

As Meta-Heurísticas que têm um comportamento positivo num dado problema de mundo-real podem não funcionar ou produzir soluções pobres na resolução de outros problemas, ou até mesmo para instâncias do mesmo problema (Chakhlevitch e Cowling, 2008). Tais limitações podem tornar-se especialmente críticas em situações onde os dados do problema e requisitos do processo podem mudar frequentemente ao longo do tempo. Isto pode fazer com que as Meta-Heurísticas se tornem dispendiosas pois devem ser mantidas de forma adequada (Chakhlevitch e Cowling, 2008).

Um dos mais antigos objetivos no campo das Meta-Heurísticas consiste em transferir uma parte do esforço de parametrização para o próprio algoritmo, dotando-o de mecanismos inteligentes para se autoadaptar ao problema/situação (Cotta *et al.*, 2008). A autoparametrização de Meta-heurísticas pode então ser efetuada com recurso a técnicas de Aprendizagem Automática, de forma a libertar o utilizador desse esforço.

A principal motivação deste trabalho surgiu da necessidade de propor e desenvolver abordagens que sejam capazes de controlar, coordenar e otimizar de forma adaptativa a resolução dos diferentes desafios existentes na otimização do problema de Escalonamento em ambientes reais de produção.

A otimização e a aprendizagem em Sistemas Multiagente são consideradas áreas de investigação promissoras mas relativamente pouco exploradas. Os agentes podem mudar o comportamento baseando-se em aprendizagem recente ou em objetivos de otimização. As estratégias de aprendizagem podem melhorar o desempenho do sistema, dotando os agentes da capacidade de aprendizagem.

Considerando que a Computação Autónoma é uma visão/desafio para o futuro, no qual os sistemas serão capazes de se autogerirem de acordo com objetivos de alto nível definidos pelo utilizador, pretende-se com este trabalho de doutoramento dar uma contribuição significativa na definição de sistemas de escalonamento com capacidades de autoparametrização.

Este trabalho de doutoramento foi realizado no âmbito do projeto de I&D *AutoDynAgents – Autonomic Agents with Self-Managing Capabilities for Dynamic Scheduling Support in a Cooperative Manufacturing System* (POCTI/EME-GIN/66848/2006), aprovado pela Fundação para a Ciência e a Tecnologia. O projeto *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 sistema *AutoDynAgents* é complexo, dinâmico e imprevisível, as questões de aprendizagem tornaram-se imprescindíveis.

Assim, surgiu a necessidade de implementação de um módulo de Auto-Otimização para a seleção de uma determinada Meta-heurística, e a especificação automática dos respetivos parâmetros, para a resolução de novos problemas de escalonamento. Este módulo requer a incorporação de técnicas de aprendizagem, de modo a dotar o sistema da capacidade de aprender com a experiência adquirida na resolução de casos anteriores similares.

1.2. Objetivos e principais contribuições

Neste trabalho de doutoramento, pretende-se investigar o desenvolvimento de sistemas inteligentes para Escalonamento assistidos por Aprendizagem, com recurso a aprendizagem por acumulação e interpretação da experiência em Escalonamento. Pretende-se também integrar várias áreas de investigação desde a Aprendizagem Automática, Computação Autónoma, Meta-Heurísticas, até à Coordenação e Aprendizagem em Sistemas Multiagente.

Identifica-se como principal objetivo deste trabalho a definição de uma plataforma baseada em Sistemas Multiagente e em técnicas inspiradas em sistemas biológicos com capacidades de aprendizagem e autogestão para a resolução de problemas de escalonamento complexos. O sistema *AutoDynAgents* deverá ser dotado da capacidade de aprendizagem para autonomamente definir os parâmetros das Meta-heurísticas a usar, através de um comportamento de Auto-Otimização.

Ao longo deste trabalho, foi efetuado o levantamento e revisão do estado da arte e descrição de:

- Abordagens de resolução de problemas de Otimização Combinatória, onde se inclui o problema de Escalonamento;
- Meta-heurísticas utilizadas ao longo deste trabalho, com foco em Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, Otimização por Colónia de Formigas, *Particle Swarm Optimization*, e Colónias de Abelhas Artificiais;
- Abordagens para a afinação de parâmetros de Meta-heurísticas;
- Características, modelos e coordenação de Sistemas Multiagente;
- Comportamentos de autogestão e desafios da Computação Autónoma;
- Aprendizagem em Sistemas Multiagente, com especial foco em Aprendizagem para a Equipa e Aprendizagem Concorrente;
- Aprendizagem para afinação de parâmetros, com ênfase nos métodos utilizados neste trabalho: *Racing* e Raciocínio baseado em Casos.

Além da apresentação do estado da arte e descrição dos aspetos referidos, identificam-se como principais contribuições deste trabalho:

- Especificação de mecanismos de aprendizagem para dotar o sistema *AutoDynAgents* com a capacidade de autoparametrização. A proposta das abordagens de aprendizagem suporta-se nas técnicas de *Racing* e Raciocínio baseado em Casos. É também proposta uma abordagem híbrida que pretende rentabilizar as vantagens de ambos os mecanismos;
- Com base nos algoritmos de aprendizagem estudados, é proposto e especificado um módulo de Auto-Otimização para integração no sistema *AutoDynagents*, de modo a que este possa resolver novas instâncias do problema de Escalonamento com o mínimo de intervenção humana;
- Realização de um estudo computacional entre abordagens de Aprendizagem para a Equipa e Aprendizagem Concorrente, para afinação de parâmetros no Sistema Multiagente *AutoDynAgents*. Este estudo permite concluir e suportar a decisão acerca de qual a abordagem a seguir;
- Realização de um estudo computacional entre as várias abordagens de aprendizagem propostas, de modo a ser possível concluir da vantagem da sua utilização.

Identifica-se também como objetivos e contribuições a escrita de artigos científicos e apresentação do trabalho em conferências internacionais. Resumindo, as contribuições científicas resultantes deste trabalho são:

- 1 publicação em revista internacional ISI (IF: 2.679);
- 3 publicações em capítulos de livros;
- 7 publicações em conferências internacionais;
- 2 co-orientações de alunos (em projetos de licenciatura e mestrado);
- Participação em 2 projetos de investigação.

Refira-se ainda a participação como co-autor em 1 publicação em revista internacional, 6 publicações em capítulos de livros, e 17 publicações em conferências internacionais.

1.3. Estrutura e organização do documento

Esta tese é composta por 10 capítulos. Após o capítulo de introdução, no capítulo 2 é abordado o problema de Escalonamento e o seu enquadramento nos problemas de Otimização Combinatória. São também descritas algumas abordagens de resolução do problema de Escalonamento usadas na literatura ao longo dos últimos anos.

No capítulo 3, efetua-se a descrição de Meta-heurísticas, com especial destaque para as usadas no âmbito deste trabalho. Descreve-se também o problema da afinação de parâmetros bem como alguns métodos para a sua resolução apresentados na literatura.

O capítulo 4 descreve os sistemas baseados em agentes e refere as motivações no uso de Sistemas Multiagente bem como os vários modelos existentes. Por apresentar alguma relevância no âmbito deste trabalho, é também apresentada uma descrição do problema de coordenação em Sistemas Multiagente, com ênfase em cooperação e competição. São ainda apresentadas algumas áreas de aplicação.

O capítulo 5 introduz o conceito de Computação Autónoma, descrevendo os comportamentos de autogestão e os aspetos da arquitetura dos sistemas autónomos bem como alguns desafios e áreas de aplicação.

No capítulo 6 discute-se a relação entre Aprendizagem Automática e Sistemas Multiagente, e referem-se alguns aspetos de aprendizagem em agentes. São apresentadas

algumas técnicas de aprendizagem aplicadas ao problema de afinação de parâmetros, com especial ênfase nos algoritmos de *Racing* e Raciocínio baseado em Casos.

No capítulo 7, é apresentado o sistema *AutoDynAgents*, nomeadamente a sua arquitetura e métodos principais. É efetuada uma descrição de todos os tipos de agentes, incluindo os agentes de autogestão, assim como dos principais módulos presentes na sua arquitetura.

No capítulo 8, são descritos os algoritmos de aprendizagem propostos para implementar a capacidade de autoparametrização. O *Racing* permite que seja efetuado um estudo entre várias combinações de parâmetros, para que seja possível determinar a melhor combinação na aplicação de uma determinada Meta-heurística. O Raciocínio baseado em Casos dá ao sistema a capacidade de evoluir e aprender com a experiência, uma vez que se baseia em casos passados e retém toda a informação da experiência para uso futuro. É ainda apresentada uma abordagem híbrida que agrega estes dois mecanismos.

O capítulo 9 apresenta o estudo computacional efetuado para a validação e análise do sistema *AutoDynAgents* com a incorporação do módulo de Auto-Otimização. O estudo computacional é dividido em duas partes: na primeira é efetuado um estudo entre Aprendizagem para a Equipa e Aprendizagem Concorrente; na segunda parte do estudo computacional valida-se a incorporação dos módulos de aprendizagem no sistema *AutoDynAgents*. Todos os resultados são validados pela análise da significância estatística. Com base nestes estudos é possível concluir sobre as vantagens de utilização dos mecanismos propostos.

As conclusões deste trabalho são apresentadas no capítulo 10, onde são descritas as principais conclusões e contribuições, sendo apresentadas direções para trabalho futuro relacionado com o trabalho desenvolvido.

Capítulo 2. O problema de Escalonamento

2.1. Introdução

O problema de Escalonamento é um dos mais antigos problemas de Otimização Combinatória para o qual têm sido desenvolvidas diversas abordagens de resolução ao longo dos anos. No entanto, muitas dessas abordagens são impraticáveis em ambientes de fabrico reais, que se distinguem por serem intrinsecamente dinâmicos, onde existem restrições complexas e perturbações inesperadas. Em grande parte 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 raramente considera estes problemas, preferindo focar-se na otimização dos planos de escalonamento estático (Ouelhadj e Petrovic, 2008).

Neste capítulo são apresentados aspetos teóricos acerca dos problemas de Otimização Combinatória, com especial realce no problema de Escalonamento. É descrito o problema de máquina única, o problema *Job-Shop*, bem como as restrições adicionais particulares da extensão *Job-Shop* Alargado (Madureira, 2003). Para finalizar o capítulo, são também descritas algumas abordagens de resolução do problema de Escalonamento usadas na literatura ao longo dos últimos anos.

2.2. Otimização Combinatória

Os problemas de Otimização Combinatória emergem quando se torna necessário selecionar, de um conjunto de dados discreto e finito, o melhor subconjunto que satisfaça determinados critérios de natureza económica-operacional.

O grande desafio da Otimização Combinatória é produzir, em tempo competitivo, soluções que sejam o mais próximo possível da(s) solução(ões) ótima(s) (soluções quase-ótimas). A dificuldade surge quando o número de alternativas possíveis (ou soluções admissíveis) se torna demasiado elevado para poder ser completamente analisado, em tempo considerado útil.

Para se ilustrar esta dificuldade, considere-se o conhecido problema do Caixeiro-Viajante (Applegate *et al.*, 2006), que consiste num conjunto de *N* cidades, numa matriz de distâncias de *NxN*, com o objetivo de percorrer todas as cidades uma única vez e voltar à cidade de partida, de forma a minimizar a distância total percorrida. Assim, as soluções admissíveis são dadas pela fórmula *(N-1)!,* o que para 5, 50, e 500 cidades obter-se-iam, respetivamente, 24, 6.08×10^{62}, e 2.44×10^{1131} percursos possíveis. Perante isto, rapidamente se percebe que analisar todas as soluções admissíveis não é viável.

Osman e Kelly (1996) definiram a Otimização Combinatória como "*o estudo matemático para encontrar uma combinação, agrupamento, ordenação ou seleção ótima de objetos discretos usualmente finito em números*".

Duma forma geral, um problema de Otimização Combinatória pode ser especificado por um conjunto de instâncias e ser representado como se mostra de seguida (Madureira, 2003; Talbi, 2009):

$$\text{P: minimização (ou maximização)f(x)}$$
$$\text{sujeito a x} \in \text{F} \subseteq \text{C}$$

Os conjuntos de decisão *C* e *F* são discretos, podendo ser definidos por um conjunto de variáveis de decisão. A cada instância é associado um espaço de solução *C*. O conjunto de soluções possíveis *F* ($F \subseteq C$) é definido pelas restrições do problema, a região de soluções impossíveis $C \backslash F$ e a função objetivo que atribui um custo real a cada solução $x \in C$, isto é, $f: C \rightarrow R$. O objectivo é então encontrar uma solução admissível $x^* \in F$, de tal forma que $f(x^*) \leq f(x')$ para qualquer $x' \in F$ onde $F \subseteq C$. A solução ótima seria um $f \in F$ tal que $\forall_{y \in F}\ C(f) \leq C(y)$.

Como Osman e Kelly (1996) referiram, "*os problemas de Otimização Combinatória são normalmente fáceis de descrever mas difíceis de resolver*".

A Teoria da Complexidade Computacional foi desenvolvida por Cook (1971) e tem o objetivo de categorizar os requisitos computacionais dos algoritmos e classificar os problemas como de fácil (*easy*) ou difícil (*hard*) resolução.

Um problema diz-se de **fácil** resolução quando é possível desenvolver um algoritmo que resolva otimamente todas as suas instâncias em tempo polinomial, limitado pelo tamanho da instância do problema, como, por exemplo, o problema de transportes, de afetação, do caminho mais curto, entre outros (Osman e Kelly, 1996).

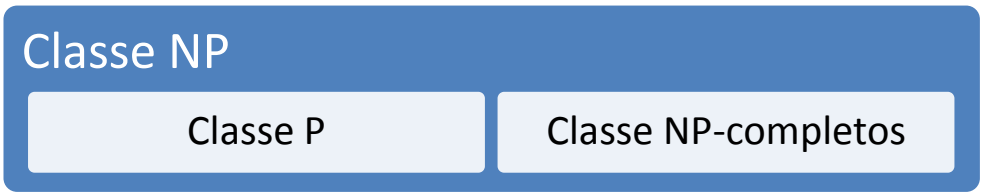

Figura 2.1 – Classes de complexidade de problemas de Otimização Combinatória (Talbi, 2009)

Um problema diz-se **difícil** se não existirem algoritmos eficientes para a sua resolução, como, por exemplo, o problema do Caixeiro-Viajante, escalonamento de tarefas, entre outros.

Um algoritmo é considerado eficiente se a sua complexidade temporal crescer de forma polinomial e não de forma exponencial com a dimensão do problema. Por **eficiência** considera-se a obtenção de soluções em tempo útil e por **eficácia** a obtenção de soluções de boa qualidade (Madureira, 2003).

Assim, como ilustrado na Figura 2.1, é possível dividir os problemas em três classes (Madureira, 2003; Talbi, 2009):

- **P**: conjunto de problemas para os quais existe um algoritmo eficiente que corre em tempo polinomial;
- **NP**: conjunto de problemas para os quais não se conhece nenhum algoritmo polinomial mas que pode ser resolvido em tempo polinomial por uma abstração algorítmica chamada "*algoritmo não determinístico*". **NP-difíceis (*NP-hard*)** são os problemas para os quais não se conhecem algoritmos eficientes de resolução, devido à sua complexidade exponencial;
- **NP-completos (*NP-complete*)**: subconjunto de problemas NP com a particularidade de que se for encontrado um algoritmo polinomial para um destes problemas, então existe um algoritmo polinomial para qualquer problema de NP.

Os resultados da Teoria da Complexidade Computacional mostram que muitos problemas de Otimização Combinatória são intratáveis, pertencendo à classe dos problemas NP-completos (*Nondeterministic polynomial-time complete*). Um algoritmo ótimo para a resolução de um problema deste tipo poderá requerer um número de passos computacionais que cresce exponencialmente com a dimensão do problema (Madureira, 2003).

2.3. Escalonamento

Os problemas de escalonamento são classificados como problemas de otimização 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 (Madureira, 2003).

Na Tabela 2.1 encontram-se algumas definições para Escalonamento, existentes na literatura.

Tabela 2.1 – Definições para Escalonamento (adaptada de (Madureira, 2003))

Autor	Definição
Baker (1974)	"O escalonamento é a atribuição de recursos no tempo para o processamento de um conjunto de tarefas"
French (1982)	"O problema de escalonamento consiste em encontrar uma sequência de passagem das tarefas pelos recursos, correspondendo a um plano exequível e ótimo em relação a um qualquer critério de otimização adotado"
Artiba e Elmaghraby (1997)	"O escalonamento consiste na determinação do sequenciamento das tarefas ou ordens de fabrico no tempo e na sua atribuição aos respetivos recursos"
Portmann (1997)	"O escalonamento pode ser descrito como a definição de tempos de início e de conclusão e a atribuição dos recursos a cada tarefa de um dado conjunto, obedecendo às várias restrições dos recursos e/ou tarefas"
Blazewicz *et al.* (2001)	"Os problemas de escalonamento podem ser definidos duma forma geral como o problema de atribuição de recursos ao longo do tempo para a realização de um conjunto de tarefas"
Pinedo (2008)	"O escalonamento é definido como a alocação de recursos a tarefas ao longo do tempo, e é um processo de tomada de decisão com o objetivo de otimizar um ou mais objetivos"

Assim, pode afirmar-se que o problema de 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). Consideram-se algumas definições para os elementos do problema de Escalonamento (Madureira, 2003):

- Uma **máquina** é caracterizada pelas suas capacidades funcionais ou qualitativas, isto é, operações que é capaz de efetuar, e capacidades quantitativas, como, por exemplo, taxa de produção, tempo de processamento, tempos de preparação, etc.);

- Uma **tarefa** é composta por um conjunto de operações que podem ser sequenciais, ou concorrentes, necessárias para a produção de um produto ou lote de produtos. A cada tarefa está associada uma gama operatória definida por um grafo de precedências de operações, o qual especifica a ordem das operações bem como, para cada operação, quais as operações que a precedem e quais as que lhe sucedem;
- Uma **operação** corresponde a um processo de uma tarefa a ser processada numa máquina. A cada operação está associado um tempo de processamento numa máquina, além da possibilidade de definir tempos de preparação, tempos de transporte e tempos de espera nessa máquina;
- Um **plano de escalonamento** é um programa de execução de tarefas, com a identificação clara da sequência de processamento das tarefas nas máquinas, bem como os tempos de início e de conclusão das operações de cada tarefa. Um plano diz-se possível, admissível ou exequível se for possível de ser totalmente cumprido obedecendo às restrições impostas ou existentes.

2.3.1. Modelos teóricos *vs.* Problemas reais

A teoria de escalonamento preocupa-se principalmente em desenvolver modelos e algoritmos para a resolução dos problemas, a partir das suas propriedades estruturais (Baker, 1974). É uma aproximação quantitativa, que tenta descrever a estrutura do problema numa forma matemática concisa e que começa com a concretização dos objetivos numa função objetivo e num conjunto de restrições (Madureira, 2003).

O problema mais simples é definido pela existência de um conjunto de tarefas uni-operação a ser executado numa única máquina (problema de máquina única, subsecção 2.3.3), com tempos de processamento e datas de entrega conhecidos e independentes da sequência em que são processadas (Madureira, 2003).

Na prática, os problemas de escalonamento são discretos, distribuídos, dinâmicos e não-determinísticos, logo muito complexos. Estes são bastante diferentes dos problemas teóricos, para os quais se destacam as seguintes características, entre outras (Pinedo, 2008):

- Os modelos teóricos não assumem alterações no conjunto de tarefas, ao longo do processo, isto é, consideram que existe um número de tarefas para escalonar e que o problema fica resolvido depois da criação dum plano de escalonamento

para esse mesmo número de tarefas. Na realidade chegam tarefas ao sistema de forma contínua e por vezes aleatória;

- Os modelos teóricos não consideram a possibilidade de reescalonamento, ou seja, na sequência da existência de acontecimentos inesperados e dinâmicos que modificam o estado do sistema, existe a necessidade de proceder a alterações mais ou menos significativas no plano escalonado anteriormente;
- Os ambientes de produção reais são mais complexos do que os teóricos, existindo restrições que podem ser dependentes dos recursos, das tarefas, ou até das datas de lançamento e de entrega;
- Nos modelos teóricos as prioridades das tarefas são fixas enquanto na prática estas podem variar ao longo do tempo;
- A maioria dos modelos teóricos não considera a disponibilidade dos recursos, assumindo que estes estão continuamente disponíveis, o que não acontece na realidade, devido a questões de manutenção ou avarias, por exemplo;
- Muitos modelos teóricos consideram apenas um único critério de otimização, enquanto na realidade se pode considerar mais do que um critério.

2.3.2. Classificação dos problemas de escalonamento

Os problemas de escalonamento podem ser classificados de várias formas. Madureira (2003) apresenta três categorias diferentes de classificação, nomeadamente, considerando a complexidade de processamento, o ambiente de escalonamento, e a variabilidade de parâmetros.

A **complexidade de processamento** permite avaliar o número de passos de processamento, isto é, o número de visitas a máquinas, e o número de máquinas associadas ao processo de produção. Segundo a complexidade de processamento, os problemas de escalonamento podem ser categorizados em problemas com tarefas uni-operação e problemas com tarefas multi-operação. Nos primeiros as tarefas são constituídas por uma única operação que é processada numa única máquina, em sistemas de máquina única ou em sistemas de máquinas paralelas. A diferença entre os dois sistemas baseia-se no facto de nas máquinas paralelas, cada tarefa ser processada numa única máquina de entre um conjunto de máquinas possíveis. Nos problemas com tarefas multi-operação, cada tarefa é constituída por um conjunto de operações processadas em máquinas diferentes, e podem ser divididos em problemas *Flow-Shop*, *Job-Shop* e *Open-Shop.* Num problema *Flow-Shop* todas as tarefas têm a mesma ordem de processamento

nas máquinas. Num problema *Job-Shop*, cada tarefa é processada numa sequência de máquinas fixa e pré-definida. Num problema *Open-Shop*, a ordem de processamento das operações nas diferentes máquinas é irrelevante.

Os problemas de escalonamento classificados segundo o **ambiente de escalonamento** podem ser divididos em problemas estáticos ou dinâmicos (French, 1982). Problemas **estáticos** são aqueles em que todas as tarefas a escalonar estão disponíveis no início do período de escalonamento e não sofrem modificações ao longo do tempo. Pelo contrário, se o número de tarefas variar ao longo do tempo, ou seja, se houver inserções ou cancelamento de tarefas, o problema é considerado como **dinâmico**.

Quanto à **variabilidade de parâmetros**, os problemas de escalonamento podem ser classificados em determinísticos e não-determinísticos. Os problemas dizem-se **determinísticos** se todos os parâmetros das tarefas são conhecidos *a priori.* Desta forma, se existirem datas de lançamento diferentes para cada tarefa, estas são conhecidas. Os problemas **não-determinísticos** são aqueles que representam um cenário real de fabrico, sujeito a fatores aleatórios de mudança que provocam alterações no estado do sistema (Blazewicz *et al.*, 2001). Assim, o plano de escalonamento precisa de ser reescalonado dinamicamente sempre que eventos inesperados ocorrem no sistema.

Por omissão, os problemas estáticos são implicitamente determinísticos, uma vez que todos os parâmetros do problema são todos conhecidos no início do período de escalonamento. A generalidade dos sistemas reais de fabrico é constituída por problemas dinâmicos não-determinísticos, isto é, o seu estado é alterado pela possibilidade de ocorrência contínua de eventos aleatórios (Madureira, 2003).

2.3.3. Problemas de máquina única

Os problemas de máquina única, isto é, que consistem em sequenciar um conjunto de tarefas numa única máquina, são problemas elementares de escalonamento, geralmente de difícil resolução (NP-difícil), em que a ordem das tarefas determina o plano de escalonamento (Madureira, 2003). São problemas determinísticos, em que as tarefas a processar numa só máquina estão todas disponíveis simultaneamente no instante de tempo zero, com os respetivos tempos de processamento e datas de entrega definidos *a priori.*

Os problemas de máquina única são principalmente úteis para a compreensão e construção de sistemas mais complexos, uma vez que para os modelar torna-se importante

conhecer o funcionamento dos seus componentes, que podem ser encarados como problemas de máquina única, procedendo-se, por vezes, à sua resolução antes de serem integrados no problema principal (Baker, 1974; Madureira, 2003; Pinedo, 2008).

Estes problemas podem ser caracterizados pelos seguintes pressupostos (Madureira, 2003):

- As tarefas são independentes e caracterizadas por tempos de processamento;
- No instante de tempo zero, a máquina está disponível para o processamento do conjunto dessas tarefas independentes;
- Os tempos de preparação das tarefas são independentes da sequência escalonada e podem ser considerados no tempo de processamento;
- Os tempos de processamento e datas de entrega são conhecidos *a priori*;
- A máquina está continuamente disponível;
- Uma vez iniciado o processamento de uma tarefa, esta é processada sem interrupções até ao fim.

Com estas condições, um conjunto de *n* tarefas num problema de máquina única pode ser sequenciado de *n!* formas, correspondendo ao número das diferentes permutações possíveis para as *n* tarefas (Madureira, 2003).

2.3.4. Escalonamento *Job-Shop* e *Job-Shop* Alargado

Num problema de escalonamento *Job-Shop* existe um conjunto de tarefas a processar num conjunto de máquinas. Cada tarefa tem uma ordem de processamento nas máquinas, ou seja, cada tarefa é composta por uma lista ordenada de operações, cada uma caracterizada pela máquina onde será processada e pelo respetivo tempo de processamento (Adams *et al.*, 1988).

Os problemas *Job-Shop* podem ser caracterizados por (Madureira, 2003):

- Um conjunto de tarefas multi-operação disponíveis para processamento no instante de tempo zero (0);
- Cada tarefa necessita de um número de operações, cada uma delas a ser processada numa máquina diferente;
- Os tempos de preparação das operações estão incluídos no tempo de processamento e são independentes da sequência escalonada;
- As máquinas são todas diferentes e estão constantemente disponíveis;

- As operações individuais não podem ser interrompidas, são processadas até ao fim;
- Não existem restrições de precedência entre operações de diferentes tarefas;
- Cada máquina só pode processar uma tarefa de cada vez e cada tarefa só pode ser processada numa máquina de cada vez;
- Os tempos de processamento e datas de entrega das tarefas são conhecidos a priori;
- Cada tarefa corresponde a um produto com estrutura de fabrico linear;
- O ambiente de fabrico é estático, uma vez que não ocorrem chegadas de novas tarefas nem ocorrem avarias nas máquinas.

A resolução destes problemas tem como objetivo a obtenção de um plano de escalonamento, isto é, uma sequência de tempos de início de cada operação, tendo em vista a concretização de determinados critérios de desempenho (funções objetivo), em que a sequência de operações nas máquinas é normalmente diferente de tarefa para tarefa (Madureira, 2003).

Num problema *Job-Shop* associam-se normalmente duas dificuldades (Madureira, 2003). A primeira está relacionada com a sua complexidade exponencial devido à explosão combinatória, uma vez que um problema de *n* tarefas em *m* máquinas tem associado $(n!)^m$ soluções possíveis. Assim, a dificuldade na sua resolução aumenta à medida que aumenta a dimensão dos problemas, sendo um problema NP-difícil uma vez que não se conhecem algoritmos eficientes para a sua resolução. A segunda dificuldade tem a ver com a diversidade de restrições a que o problema pode estar sujeito, por exemplo, a existência de datas de lançamento e de entrega das tarefas, bem como a disponibilidade dos recursos, restrições de precedência de tarefas, entre outras.

Os problemas considerados neste trabalho contêm algumas restrições adicionais ao *Job-Shop*, sendo importantes para uma representação mais real do processo de fabrico.

Os problemas *Job-Shop* Alargado consistem em problemas *Job-Shop* com restrições adicionais, de modo a representar melhor a realidade industrial (Madureira, 2003). As restrições adicionais consideradas por este tipo de problemas são:

- A existência de diferentes datas de lançamento e de entrega para cada tarefa;
- A possibilidade da existência de prioridades associadas às tarefas;
- A possibilidade de nem todas as máquinas serem usadas por todas as tarefas;

- A possibilidade de uma tarefa ter mais do que uma operação a ser realizada na mesma máquina;
- A possibilidade de processamento simultâneo de operações pertencentes a uma mesma tarefa;
- A existência de máquinas alternativas, idênticas ou não, para o processamento de uma operação;

Nenhuma tarefa inicia o seu processamento antes da respetiva data de lançamento, e a sua conclusão não deve ultrapassar a data de entrega.

2.4. Abordagens de resolução

Várias técnicas de otimização têm sido usadas para resolver o problema de escalonamento, tais como Regras de Prioridade, Meta-heurísticas, Sistemas Multiagente, Sistemas baseados em Conhecimento, entre outras (Ouelhadj e Petrovic, 2008; Pinedo, 2008).

As **Regras de Prioridade** (*Dispatching Rules*) são procedimentos simples, usados frequentemente em escalonamento, para a determinação da sequência segundo a qual as operações devem ser realizadas, tendo em vista determinados critérios de desempenho (Madureira, 2003). No entanto, a sua principal desvantagem recai sobre o facto de as soluções serem muitas vezes pobres devido à sua natureza míope (Ouelhadj e Petrovic, 2008). Referem-se algumas Regras de Prioridade (Baker, 1974; Madureira, 2003):

- **EDD (*Earliest Due Date*)** onde é dada prioridade à tarefa com menor data de entrega (*due date*, em inglês), i.e. as tarefas são ordenadas por ordem crescente das suas datas de entrega;
- **SPT (*Shortest Processing Time*)** onde é dada prioridade à tarefa com menor tempo de processamento na máquina em consideração;
- **LPT (*Longest Processing Time*)** que, ao contrário da SPT, dá prioridade à tarefa com maior tempo de processamento em determinada máquina;
- **FCFS (*First Come First Served*)** onde a primeira tarefa a ser executada é aquela que estiver primeiramente disponível para processamento;
- **RND (*Random*)** que ordena as tarefas aleatoriamente;

- **REDD (*Randomized Earliest Due Date*)** que permite a introdução de alguma perturbação numa sequência EDD pela troca de pares de tarefas definidas duma forma aleatória (Madureira, 2003);
- ***SeqNivel***, proposta por Madureira (2003), em que a sequência de operações é definida pela "*ordenação não-decrescente dos níveis a que pertencem as operações, dando assim prioridade às operações das tarefas que intervêm mais cedo*".

Nos últimos anos, as **Meta-heurísticas** têm sido usadas com sucesso para resolver o problema de escalonamento (Ouelhadj e Petrovic, 2008). Estas técnicas são heurísticas de alto nível que guiam o processo de procura de forma a escapar aos ótimos locais, comuns em heurísticas de pesquisa local (Glover e Laguna, 1998; Pham e Karaboga, 1998; Reeves, 1993). As heurísticas de pesquisa local são métodos baseados na pesquisa de vizinhanças. No algoritmo de Pesquisa Local, a procura começa a partir de uma dada solução, e tenta iterativamente mover-se para uma solução melhor, de acordo com a vizinhança da solução atual, através do uso de operadores. Este processo para quando não for possível encontrar, na vizinhança, soluções melhores do que a atual, o que representa um ótimo local. As Meta-heurísticas, tais como Pesquisa Tabu, *Simulated Annealing*, Algoritmos Genéticos, entre outras, que serão explicadas em maior detalhe no Capítulo 3, melhoram o processo de procura de forma a permitir escapar a ótimos locais, através da aceitação de soluções piores, ou através da geração de boas soluções iniciais para a pesquisa, o que se revela mais inteligente do que apenas fornecer soluções iniciais aleatórias.

Os **Sistemas Multiagente** são sistemas onde existe um conjunto de agentes que interagem entre si e tentam chegar a uma solução para resolver o problema em questão, de forma cooperativa ou competitiva. Têm vindo a ser cada vez mais usados para a resolução dos mais variados tipos de problemas (Bellifemine *et al.*, 2007; Weiß, 1999; Wooldridge, 2002), entre eles os problemas de escalonamento. Uma vez que o âmbito deste trabalho engloba este tipo de sistemas, pode ser encontrada uma descrição mais alargada no Capítulo 4.

Os **Sistemas baseados em Conhecimento** focam-se em capturar o conhecimento especializado de peritos num determinado domínio e usar mecanismos de inferência para derivar conclusões ou recomendações em relação às ações a tomar. Estes sistemas possuem o potencial para automatizar e incorporar o raciocínio dos peritos e o conhecimento heurístico para executar sistemas de escalonamento de produção. No

entanto, falta-lhes normalmente a capacidade para otimizar o sistema e requerem um esforço considerável para os construir e manter. Têm como objetivo gerar soluções exequíveis de acordo com o domínio do problema, mas, em termos de eficácia na capacidade de tomada de decisão, os Sistemas baseados em Conhecimento são limitados pela qualidade e integridade do conhecimento específico de domínio (Ouelhadj e Petrovic, 2008).

2.5. Sumário

Neste capítulo foram descritos os problemas de Otimização Combinatória integrando alguns aspetos sobre a Teoria da Complexidade Computacional. Uma vez que o âmbito deste trabalho recai sobre o problema do Escalonamento, foi apresentada uma comparação entre os modelos teóricos e os problemas reais, e classificou-se os problemas de escalonamento de acordo com a complexidade de processamento, o ambiente de escalonamento, e a variabilidade de parâmetros. Os problemas de escalonamento descritos foram os problemas de máquina única, o escalonamento *Job-Shop*, e a extensão *Job-Shop Alargado*, uma vez que este último contém restrições importantes para uma representação mais rigorosa da realidade industrial.

Por fim, foram apresentadas algumas das técnicas mais usadas na resolução do problema de Escalonamento, nomeadamente as Regras de Prioridade, as Meta-heurísticas, os Sistemas Multiagente, e os Sistemas baseados em Conhecimento.

Capítulo 3. As Meta-heurísticas e o problema da afinação de parâmetros

3.1. Introdução

A computação de soluções ótimas é intratável para muitos problemas de otimização de importância industrial e científica (Bartz-Beielstein *et al.*, 2004). Assim, a adaptação de ideias oriundas de várias áreas, principalmente com base na natureza, resultou no desenvolvimento de Meta-heurísticas de inspiração natural, que consistem em métodos heurísticos de pesquisa para solucionar problemas complexos de Otimização Combinatória. Este tipo de Meta-heurísticas ganhou muita popularidade ao longo das últimas duas décadas (Birattari, 2009; Gonzalez, 2007; Talbi, 2009) e cada vez mais as empresas usam estas técnicas para resolver problemas de otimização em vários domínios. A disponibilidade crescente do poder computacional de baixo custo é um dos fatores motivadores da sua utilização por parte das empresas, que em muitos aspetos da sua atividade (incluindo a especificação, o planeamento, o inventário, a distribuição, a logística, a gestão, etc.) beneficiam largamente da otimização apropriada. Neste contexto, as Meta-heurísticas são uma solução particularmente apelativa pois garantem boas soluções mesmo quando existem poucos recursos computacionais.

As Meta-heurísticas têm como objetivo guiar e melhorar o processo de pesquisa de modo a superar as soluções ótimas locais, que constituem uma limitação no algoritmo de Pesquisa Local (ver subsecção 3.2.1), e obter soluções com melhor qualidade, o mais próximas possível da solução ótima, em tempos de computação aceitáveis (Gonzalez, 2007; Madureira, 2003; Talbi, 2009). A principal vantagem das Meta-heurísticas baseia-se em serem fáceis de implementar sendo possível produzir uma versão tipo "*quick-and-dirty*" de uma Meta-heurística para um determinado conjunto de problemas em pouco tempo. Tais implementações são tipicamente capazes de um desempenho razoável. Contudo, quando se pretende resultados próximos do ótimo, são necessárias decisões de especificação cautelosas e afinações precisas (Birattari, 2009).

Embora a sua importância seja bem reconhecida na literatura, o processo de afinação de parâmetros de Meta-heurísticas tem recebido relativa pouca atenção e só recentemente foi proposta uma metodologia bem estabelecida para o tratar de forma eficiente e eficaz (Birattari, 2009). De facto, a afinação de parâmetros ainda é muitas vezes

realizada sem a utilização de procedimentos automáticos, o que acarreta um número de desvantagens tais como, e.g., reduzida replicabilidade do processo, pouca fiabilidade e erros derivados de processos de ajuste tediosos (Birattari, 2009; Hamadi *et al.*, 2012; Hutter *et al.*, 2006a).

Neste capítulo, será efetuada a descrição do funcionamento básico de algumas Meta-heurísticas, com especial destaque para as usadas no âmbito deste trabalho, nomeadamente a Pesquisa Tabu, o *Simulated Annealing*, os Algoritmos Genéticos, a Otimização por Colónia de Formigas, o *Particle Swarm Optimization*, e a Colónia de Abelhas Artificiais. Será ainda descrito com detalhe o problema da afinação de parâmetros bem como referidos alguns métodos, apresentados na literatura, para a sua resolução.

3.2. Meta-heurísticas

O termo "Meta-heurística" foi introduzido por Fred W. Glover em 1986 (Glover, 1986) e resulta da junção das palavras gregas *heuriskein*, que significa descobrir, e *meta*, que significa "*metodologia de nível superior*" (Talbi, 2009). A palavra heurística, por si só, é a arte de descobrir novas estratégias ou regras para a resolução de problemas. Ao contrário dos métodos exatos, que quando aplicáveis, garantem a solução ótima, as Meta-heurísticas permitem a resolução de problemas de grandes dimensões através da obtenção de soluções satisfatórias em tempos de execução aceitáveis. No entanto, não há a garantia da obtenção da solução ótima (Madureira, 2003; Talbi, 2009). Sendo assim, as Meta-heurísticas consistem em métodos iterativos ou recursivos com o objetivo de obter soluções o mais próximo possível do ótimo global para um determinado problema. Assumindo que todas as soluções estão relacionadas entre si, podendo-se, a partir de cada uma, alcançar um conjunto de soluções para o problema, chama-se de trajetória no espaço das soluções ao caminho percorrido pelo método até encontrar a solução.

Existem diversos conceitos comuns aos vários tipos de Meta-heurísticas (Gonzalez, 2007; Madureira, 2003; Pirlot, 1996; Talbi, 2009). A solução inicial pode ser gerada aleatoriamente ou através de heurísticas construtivas. A função objetivo depende do problema a resolver, pode ser de maximização ou minimização, e é um dos elementos mais importantes na implementação de uma Meta-heurística, pois define o objetivo a atingir e orienta o processo de pesquisa à procura de boas soluções. A estrutura da vizinhança ou população é igualmente importante pois define o espaço de procura de soluções. Para critério de paragem é normalmente usado o número máximo de iterações ou um número de

iterações sem melhoria na melhor solução, podendo também ser usado um determinado limite de tempo computacional.

Mediante a necessidade de obtenção de uma solução para um problema, deve ser escolhida uma determinada Meta-heurística. Sendo esta escolha considerada difícil, deve ser realizado um estudo do tipo de problema e da compreensão do modo de funcionamento da técnica que se pretende usar. É também difícil eleger uma Meta-heurística como sendo a melhor de todas, pois depende do problema em causa e cada uma apresenta vantagens e desvantagens. O processo de seleção deve ter por base os resultados que se querem obter, podendo considerar-se como critérios a eficiência, a eficácia, etc. (Pereira, 2009).

As Meta-heurísticas têm sido usadas na resolução de problemas em vários domínios de aplicação, sendo especialmente eficientes e eficazes na resolução de problemas de grande dimensão e complexos (Glover e Kochenberger, 2003; Talbi, 2009). Luke (2012) referiu que estas técnicas são especialmente úteis na resolução de problemas para os quais não se sabe como descobrir a solução ótima, onde há pouco conhecimento heurístico, e onde o uso de força bruta não é aconselhado porque a complexidade da pesquisa o torna proibitivo em termos de tempo computacional.

Podem ser encontradas na literatura várias abordagens relativamente à abrangência das Meta-heurísticas, mas genericamente pode dizer-se que se dividem em três categorias (Glover e Kochenberger, 2003; Gonzalez, 2007; Madureira, 2003; Talbi, 2009): baseadas no algoritmo de Pesquisa Local (e.g. Pesquisa Tabu, *Simulated Annealing*, GRASP, entre outras), as evolucionárias (e.g. Algoritmos Genéticos, Algoritmos Meméticos, Evolução Diferencial), ou as pertencentes à *Swarm Intelligence* (tais como Otimização por Colónia de Formigas, *Particle Swarm Optimization,* Colónia de Abelhas Artificiais, entre outras).

A *Swarm Intelligence* engloba os algoritmos inspirados no comportamento coletivo de espécies animais tais como formigas, abelhas, pássaros, peixes, etc., e baseia-se principalmente na cooperação inerente à comunicação indireta em tais espécies (Karaboga, 2005; Karaboga e Akay, 2009; Talbi, 2009). A auto-organização e a divisão de trabalho são dois conceitos fundamentais para se obter os comportamentos de *Swarm Intelligence*. Para isso basta que os indivíduos sigam determinadas regras.

3.2.1. Pesquisa Local

O algoritmo de Pesquisa Local, também conhecido por *Hill Climbing*, é talvez o mais antigo e mais simples método meta-heurístico (Luke, 2012; Pirlot, 1996; Talbi, 2009). Explora o pressuposto de que soluções similares tendem a comportar-se da mesma forma. Assim, pequenas modificações resultam geralmente em mudanças pequenas mas boas em termos de qualidade, o que nos permite "*subir a colina*" de qualidade até às boas soluções.

A Pesquisa Local é a base das Meta-heurísticas uma vez que muitas dessas técnicas se baseiam na combinação de pesquisas aleatórias com pesquisas locais, entre outros comportamentos (Luke, 2012; Pirlot, 1996). Um algoritmo genérico de Pesquisa Local é apresentado no Algoritmo 3.1. Este algoritmo começa com uma dada solução inicial, gerada aleatoriamente ou gerada através de uma regra de prioridade, e, a cada iteração, substitui a solução atual por um vizinho que melhora a função objetivo. A pesquisa para quando todos os vizinhos candidatos são piores do que a solução atual, o que significa que foi atingido um mínimo local. Para grandes vizinhanças, as soluções candidatas devem ser um subconjunto da vizinhança, para acelerar o processo de pesquisa (Talbi, 2009). Além da definição da solução inicial e do tamanho da vizinhança, a implementação de um algoritmo de Pesquisa Local tem de se preocupar com a estratégia de seleção de vizinhos, que poderá influenciar a trajetória das soluções.

Geralmente, a Pesquisa Local é um método fácil de implementar e apresenta soluções de qualidade elevada muito rapidamente. No entanto, a grande desvantagem é que converge para ótimos locais. Além disso, o algoritmo pode ser sensível à solução inicial (Madureira, 2003).

Este algoritmo funciona bem se não houver muitos ótimos locais no espaço de pesquisa ou então se a qualidade dos diferentes ótimos locais for mais ou menos similar (Talbi, 2009). No entanto, esta é uma questão que é muito difícil de saber à partida.

Recentes aplicações do algoritmo de Pesquisa Local à resolução do problema de escalonamento podem ser encontradas em (Essafi *et al.*, 2008; Mika *et al.*, 2011; Moslehi e Mahnam, 2011; Tseng e Lin, 2009; Vela *et al.*, 2010; Watson e Beck, 2008; Zhang e Wu, 2012).

Algoritmo 3.1 – Pesquisa Local

```
    Entrada: SolInicial, CritParagem // Solução inicial e critério de paragem (n° máximo de iterações)
    Saída: SolucaoAtual // Solução correspondente ao ótimo local
1   NumIteracoes ← 0;
2   SolucaoAtual ← SolInicial;
3   Repete
4           ListaVizinhos ← gerarVizinhos(SolucaoAtual); // Gerar vizinhança da solução atual
5           // Variável para controlar se encontrou um vizinho melhor que a solução atual
6           ExisteVizinhoMelhor ← false;
7           Para cada Vizinho_i ∈ ListaVizinhos Faz // Avalia todos os vizinhos gerados
8                   Se qualidade(Vizinho_i) > qualidade(SolucaoAtual) Então
9                           // Se o vizinho é melhor que a solução atual, atualiza as variáveis
10                          SolucaoAtual ← Vizinho_i;
11                          ExisteVizinhoMelhor ← true;
12                  Fim
13          Fim
14  // Repete o ciclo enquanto não atinge o critério de paragem e encontrar um vizinho melhor
15  Enquanto NumIteracoes++ < CritParagem E ExisteVizinhoMelhor;
16  Devolver SolucaoAtual;
```

3.2.2. Pesquisa Tabu

A Pesquisa Tabu foi introduzida por Fred W. Glover (1986) e tornou-se muito popular na década de 90, sendo hoje em dia uma das Meta-heurísticas mais utilizadas. Consiste numa estratégia de Pesquisa Local com o objetivo principal de escapar aos mínimos locais, uma vez que, ao contrário da Pesquisa Local, aceita soluções piores quando a vizinhança não apresenta melhorias (Glover e Kochenberger, 2003; Gonzalez, 2007; Pirlot, 1996; Talbi, 2009).

A principal característica da Pesquisa Tabu é o uso de memória, que guarda informação relativa ao processo de pesquisa de soluções, sendo esta uma semelhança com o comportamento humano, na medida em que o passado influência, de forma determinante, o processo de pesquisa de novas soluções, tal como o passado de um ser humano define as suas decisões futuras (Luke, 2012; Madureira, 2003; Talbi, 2009). Desta forma, usando memória, é possível abandonar ótimos locais consentindo uma deterioração no valor da qualidade de uma solução, desde que essa nova solução não tenha sido já visitada anteriormente. O algoritmo verifica se essa solução já foi visitada através da utilização de uma lista tabu que armazena a informação referente às soluções já visitadas. Assim, este processo de pesquisa por melhores soluções é feita através de movimentos, os quais são realizados para as melhores soluções encontradas e seguindo certas regras. Alguns movimentos são classificados como proibidos, ou tabu, o que previne a ocorrência de ciclos,

mínimos locais, e possibilita encontrar ótimos globais (Glover e Kochenberger, 2003; Madureira, 2003; Pirlot, 1996; Talbi, 2009). Esta Meta-heurística, como descrito no Algoritmo 3.2, parte de uma solução inicial (gerada aleatoriamente ou através de uma regra de prioridade) e escolhe para solução seguinte a melhor solução da vizinhança atual, ou duma subvizinhança, caso a vizinhança seja demasiado extensa para poder ser eficientemente explorada pelo algoritmo. O melhor vizinho é então adicionado à lista tabu e, no caso de esta ficar cheia, o registo mais antigo da lista é removido. Depois de um número predeterminado de ciclos, o algoritmo devolve a melhor solução encontrada.

Algoritmo 3.2 – Pesquisa Tabu

```
    // Solução inicial, critério de paragem (número máximo de iterações) e tamanho da lista tabu
    Entrada: SolInicial, CritParagem, TamanhoListaTabu
    Saída: MelhorSolucao // Melhor solução encontrada pelo algoritmo
1   NumIteracoes ← 0;
2   MelhorSolucao ← SolInicial; // Inicializa a melhor solução igual à solução inicial
3   // Cria lista tabu de tamanho igual ao parâmetro de entrada
4   ListaTabu ← Array(TamanhoListaTabu);
5   // Enquanto o critério de paragem não é atingido
6   Enquanto NumIteracoes++ < CritParagem Faz
7           ListaVizinhos ← gerarVizinhos(SolucaoAtual); // Gera a vizinhança da solução atual
8           // Obtém o melhor vizinho da vizinhança que não pertence à lista tabu
9           MelhorVizinho ← obterMelhorVizinho(ListaVizinhos, ListaTabu);
10          // Adiciona o melhor vizinho à lista tabu removendo o mais antigo se estiver cheia
11          ListaTabu.adicionar(MelhorVizinho);
12          Se qualidade(MelhorVizinho) > qualidade(MelhorSolucao) Então
13                  // Atualiza a melhor solução se o melhor vizinho tiver qualidade superior
14                  MelhorSolucao ← MelhorVizinho;
15          Fim
16  Fim
17  Devolver MelhorSolucao;
```

A lista tabu consiste numa lista com o objetivo de guardar os pares de soluções envolvidos nos últimos *k* movimentos. Se um par de soluções estiver na lista tabu, então esse movimento não é permitido, sendo possível, além da prevenção de ciclos, explorar também outros espaços de procura de soluções. Estes pares de soluções são colocados na lista tabu consoante são visitadas novas soluções. Normalmente, a lista tabu tem uma dimensão fixa e, uma vez atingido o limite, o registo mais antigo é eliminado. Com este procedimento não se garante totalmente a ausência de ciclos, sendo também necessária a definição de um critério de paragem baseado num número total pré-definido de iterações, momento em que o algoritmo termina (Glover e Kochenberger, 2003; Luke, 2012; Talbi, 2009).

Na especificação da Pesquisa Tabu podem ser introduzidos mecanismos de intensificação e diversificação, que permitem uma análise sistemática do domínio das soluções, através do registo do percurso efetuado pelas últimas iterações. Permitem ainda a análise de soluções na vizinhança da solução atual. As soluções obtidas em cada iteração são também analisadas, de modo a extrair características que possam ser comuns a boas soluções e, assim, definir as "direções de procura" que poderão conduzir às melhores soluções (Madureira, 2010).

A intensificação está centrada na definição do critério de escolha da solução seguinte com base nas melhores soluções encontradas, registadas nas estruturas de memória, i.e. a procura é feita na vizinhança daquelas soluções, mas são também criadas novas vizinhanças, que resultam da combinação de boas características das melhores soluções obtidas. A diversificação é uma fase complementar à intensificação que consiste em incentivar a pesquisa em regiões ainda não exploradas, penalizando as características comuns das soluções visitadas mais recentemente (Madureira, 2010).

Para que um algoritmo de Pesquisa Tabu possa ser executado, é necessário indicar uma solução inicial, especificar uma estrutura de vizinhança, e considerar que pode ser também necessária uma estrutura de subvizinhança, em vez de uma vizinhança total para cada solução, que pode ser a seleção aleatória de uma certa quantidade de soluções vizinhas da totalidade da vizinhança. Além disso, é necessário especificar o tamanho e conteúdo da lista tabu (Gonzalez, 2007; Madureira, 2003; Pirlot, 1996; Talbi, 2009). A definição do conteúdo da lista tabu é importante uma vez que a informação guardada pode acabar por se tornar demasiado restritiva, pois é possível excluir mais soluções do que aquelas que se pretendem, incluindo soluções que ainda não foram sequer visitadas (Glover e Kochenberger, 2003; Luke, 2012; Madureira, 2003; Talbi, 2009). Para contornar este problema, em vez de se guardar a descrição completa das soluções, pode-se guardar apenas uma ou duas características do movimento ocorrido, por exemplo, definir as modificações que transformam a solução atual na nova solução (Luke, 2012). Aplicações desta Meta-heurística à resolução de problemas de escalonamento podem ser encontradas, por exemplo em (Li *et al.*, 2010; Li *et al.*, 2011; Liang e Tao, 2011; Madureira *et al.*, 2009a; Pereira e Madureira, 2013; Pereira *et al.*, 2013a; Prot e Bellenguez-Morineau, 2012; Vilcot e Billaut, 2008; Wang e Tang, 2009).

3.2.3. *Simulated Annealing*

O *Simulated Annealing* é um algoritmo relativamente antigo, cuja origem se pode situar na década de 1980 nos trabalhos de Kirkpatrick *et al.* (1983) e Černý (1985), e tem ligações à termodinâmica e à metalurgia através do trabalho de Metropolis *et al.* (1953).

A motivação original deste algoritmo surgiu com base no processo em que o metal em fusão é arrefecido lentamente de modo a solidificar numa forte estrutura cristalina. A robustez desta estrutura depende da taxa de arrefecimento do metal. Se a temperatura inicial não for suficientemente alta ou se o arrefecimento for demasiado rápido, os átomos não ficam juntos e surgem imperfeições, resultando em metais quebradiços. Se a temperatura for diminuída lentamente os átomos têm tempo suficiente para assentar e resultar numa estrutura forte (Glover e Kochenberger, 2003; Luke, 2012; Pirlot, 1996; Talbi, 2009). A Tabela 3.1 ilustra a analogia entre os sistemas físicos e o algoritmo de *Simulated Annealing*.

Tabela 3.1 – Analogia entre um sistema físico e o algoritmo de *Simulated Annealing* (Talbi, 2009)

Sistema físico	*Simulated Annealing*
Estado do sistema	Solução
Energia	Função objetivo
Estado fundamental	Solução ótima global
Estado meta-estável	Ótimo local
Temperatura	Parâmetro T

O *Simulated Annealing* é um algoritmo estocástico que, dentro de determinadas condições, permite a degradação da solução atual através de movimentos para soluções de pior qualidade, de modo a escapar ao mínimo local e a atrasar a convergência do algoritmo. É a combinação da geração aleatória e do processo de melhoria iterativa a partir de comparações e probabilidades (Glover e Kochenberger, 2003; Talbi, 2009). No início do processo iterativo, quase todos os movimentos são aceites, isto é, são aceites todas as substituições da solução atual por uma nova solução da sua vizinhança, escolhida aleatoriamente, o que permite explorar o espaço das soluções. À medida que o algoritmo vai evoluindo, a temperatura vai sendo diminuída gradualmente (arrefecimento), tornando o algoritmo progressivamente mais seletivo na aceitação de novas soluções, até que, no final, apenas são aceites os movimentos que melhoram a solução atual.

O *Simulated Annealing* encontra-se descrito no Algoritmo 3.3 e, ao contrário da Pesquisa Tabu, não tem memória, uma vez que não utiliza nenhuma informação adquirida ao longo do processo de pesquisa.

Assim, começa-se por gerar uma solução inicial *S*, que pode ser gerada aleatoriamente ou construída através duma heurística, e inicializa a temperatura *T* com um dado valor. Enquanto o critério de paragem, ou número máximo de iterações, 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 sempre que 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, geralmente com base na distribuição de Boltzmann (equação (3.1)), onde *f(S')* e *f(S)* representam o valor objetivo de *S'* e *S*, respetivamente (Blum e Roli, 2003; Glover e Kochenberger, 2003; Luke, 2012; Pirlot, 1996; Talbi, 2009).

Algoritmo 3.3 – *Simulated Annealing*

```
    // Solução inicial, critério de paragem, temperatura inicial, e n° de iterações à mesma temperatura
    Entrada: SolInicial, CritParagem, TempInicial, K
    Saída: MelhorSolucao // Melhor solução encontrada pelo algoritmo
1   NumIteracoes ← 0;
2   // Inicializa a melhor solução e a solução atual iguais à solução inicial
3   MelhorSolucao ← SolucaoAtual ← SolInicial;
4   Temp ← TempInicial;
5   Enquanto NumIteracoes++ < CritParagem Faz
6           i ← 0; // Inicializa contador de iterações à mesma temperatura
7           Repete
8                   Vizinho ← gerarVizinho(SolucaoAtual); // Gerar um vizinho da solução atual
9                   Se qualidade(Vizinho) > qualidade(SolucaoAtual) Então
10                          // Se o vizinho for melhor do que a solução atual
11                          SolucaoAtual ← Vizinho; // Atualiza a solução atual
12                          Se qualidade(SolucaoAtual) > qualidade(MelhorSolucao) Então
13                                  // Atualiza a melhor solução se a solução atual for melhor
14                                  MelhorSolucao ← SolucaoAtual;
15                          Fim
16                  Senão // Se o vizinho não for melhor do que a solução atual
17                          Se rand() < calcularProb(SolucaoAtual,Vizinho,Temp) Então
18                                  // Aceita o vizinho com base na probabilidade (equação (3.1)
19                                  SolucaoAtual ← Vizinho;
20                          Fim
21                  Fim
22          Enquanto i++ < K; // número de ciclos à mesma temperatura
23          atualizarTemperatura(Temp); // Atualizar a temperatura com base na equação (3.2)
24  Fim
25  Devolver MelhorSolucao;
```

$$P = e^{\left(-\frac{f(S')-f(S)}{T}\right)} \tag{3.1}$$

À medida que o algoritmo progride, a probabilidade de aceitação de movimentos para soluções piores vai diminuindo, tal como ilustrado na Figura 3.1. Quanto mais alta for a temperatura, maior será a probabilidade de aceitação de um movimento "mau". Um movimento que permite a melhoria da solução atual é sempre aceite.

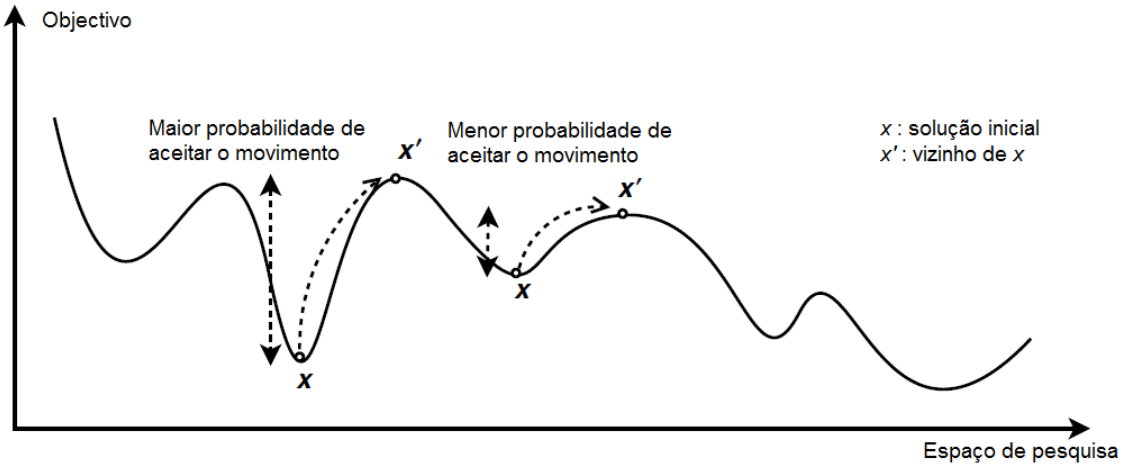

Figura 3.1 – *Simulated Annealing* a escapar aos ótimos locais (adaptada de (Talbi, 2009))

Além da temperatura inicial, o arrefecimento é essencial para o desempenho do algoritmo, sendo normalmente efetuado de *k* em *k* iterações. Normalmente, é aplicado um fator de redução de temperatura (α) de modo a reduzir a temperatura de forma constante, como ilustrado na equação (3.2), onde *k* representa o número da iteração. Quanto mais lento for o arrefecimento, maior será a diversidade da pesquisa no espaço de soluções (Blum e Roli, 2003; Glover e Kochenberger, 2003; Talbi, 2009).

$$T_{k+1} = \alpha T_k \tag{3.2}$$

Para que este algoritmo funcione corretamente, além da necessidade de definição da solução inicial, é também necessário especificar o número de iterações realizadas à mesma temperatura, a temperatura inicial e a taxa de arrefecimento (Gonzalez, 2007; Talbi, 2009). Aplicações desta Meta-heurística à resolução de problemas de escalonamento podem ser encontradas, e.g. em (Chen e Shahandashti, 2009; Goswami *et al.*, 2011; Lin *et al.*, 2009; Naderi *et al.*, 2010; Pereira e Madureira, 2013; Pereira *et al.*, 2013a; Song *et al.*, 2012; Wang *et al.*, 2011a; Zhang e Wu, 2010).

3.2.4. Algoritmos Genéticos

Nos princípios da década de 1970, John Holland, em conjunto com os seus alunos e colegas da Universidade de Michigan nos EUA, desenvolveu pesquisas e estudos baseados na área da seleção natural de espécies, tendo alcançado um modelo formal designado por Algoritmos Genéticos (Holland, 1975). Estes estudos tiveram fundamento na genética e na ideia apresentada num livro escrito por Charles Darwin (1859) que descrevia a teoria da origem das espécies. Neste livro, Darwin descreve competições entre indivíduos de cada espécie animal e refere que apenas os mais aptos conseguem prevalecer, sendo a natureza responsável por efetuar esta seleção natural, suportando assim a teoria de evolução das espécies. Nos anos 1980, David Goldberg, aluno de Holland, publicou as primeiras aplicações bem-sucedidas destes algoritmos (Goldberg, 1989). Desde então, os Algoritmos Genéticos têm sido aplicados com sucesso nos mais diversos tipos de problemas de otimização (Glover e Kochenberger, 2003; Madureira, 2003; Talbi, 2009).

Na Tabela 3.2 encontram-se descritos os principais termos usados nos Algoritmos Genéticos.

Tabela 3.2 – Termos usados nos Algoritmos Genéticos (Luke, 2012)

Termo	Definição
Indivíduo	Uma solução candidata
Progenitor e descendente	Um descendente é um indivíduo resultante do cruzamento de duas soluções candidatas (progenitores)
População	Conjunto de indivíduos
Aptidão (*fitness*)	Qualidade de um indivíduo
Seleção	Escolher indivíduos com base na sua aptidão
Cruzamento	Escolher dois progenitores e trocar secções entre eles para, normalmente, produzir dois descendentes. Isto é muitas vezes pensado como reprodução “sexual”
Mutação	Ajuste simples, muitas vezes pensado como reprodução “assexuada”
Reprodução	Produção de um ou mais descendentes de uma população de progenitores através de um processo iterativo de seleções, cruzamentos e mutações
Geração	População produzida em cada iteração
Genótipo ou genoma	Estrutura de dados de um indivíduo
Cromossoma	Um genótipo na forma de um vetor de dimensão fixa (solução codificada)
Gene	Uma determinada posição num cromossoma
Alelo	Um valor particular de um gene

A principal diferença entre estes algoritmos e outras Meta-heurísticas, tais como a Pesquisa Tabu e o *Simulated Annealing,* é o facto dos Algoritmos Genéticos lidarem com conjuntos de soluções (populações), em vez de lidarem com uma solução isolada. A exploração das soluções não é efetuada apenas à vizinhança de uma solução isolada, sendo pelo contrário efetuada uma exploração da vizinhança de uma população inteira.

Uma particularidade dos Algoritmos Genéticos é o facto de, usualmente, os seus operadores genéticos não atuarem diretamente no espaço de soluções. Em vez disso, as soluções devem ser codificadas como *strings*, de comprimento e alfabeto finito, conhecidas como cromossomas (Luke, 2012; Pirlot, 1996).

Algoritmo 3.4 – Algoritmo Genético

```
    // Tamanho da população, critério de paragem, taxa de cruzamento e taxa de mutação
    Entrada: TamPop, CritParagem, TxCruz, TxMut
    Saída: MelhorIndividuo // Melhor solução encontrada pelo algoritmo
1   NumGeracoes← 0;
2   MelhorIndividuo ← ∅;
3   Populacao ← geraPopulacao(TamPop); // Gera uma população com TamPop indivíduos
4   Enquanto NumGeracoes++ < CritParagem Faz // Enquanto o critério de paragem não é atingido
5           Para cada Individuo_i ∈ Populacao Faz
6                   // Percorre todos os indivíduos da população para guardar o melhor
7                   Se qualidade(Individuo_i) > qualidade(MelhorIndividuo) Então
8                           // Atualiza o melhor indivíduo encontrado
9                           MelhorIndividuo ← Individuo_i;
10                  Fim
11          Fim
12          i ← 0; // Inicializa contador auxiliar
13          NovaPop ← ∅; // Inicializa nova população
14          Repete
15                  ProgenitorA ← selecao(Populacao); // Seleciona um indivíduo da população
16                  ProgenitorB ← selecao(Populacao); // Seleciona outro indivíduo da população
17                  // Aplica cruzamento entre dois indivíduos, com base na taxa de cruzamento
18                  {DescendenteA, DescendenteB} ← cruzamento(ProgenitorA,ProgenitorB,TxCruz);
19                  // Aplica a mutação aos descendentes consoante a taxa de mutação
20                  mutacao(DescendenteA,TxMut);
21                  mutacao(DescendenteB,TxMut);
22                  // Adiciona os descendentes à nova população
23                  NovaPop.adicionar(DescendenteA);
24                  NovaPop.adicionar(DescendenteB);
25          Enquanto i++ < TamPop/2; // Percorre metade da população
26          Populacao ← NovaPop; // Atualiza a população
27  Fim
28  Devolver MelhorIndividuo;
```

Um Algoritmo Genético (Algoritmo 3.4) inicia-se com uma população de *N* indivíduos (soluções) e evolui produzindo, em cada iteração, uma nova população do mesmo tamanho. Em cada nova geração, os indivíduos da população (progenitores) são selecionados e cruzados entre si, originando novos indivíduos (descendentes) que normalmente substituem os ascendentes na população atual. Após serem escolhidos os pares de indivíduos progenitores, procede-se à operação de cruzamento mediante uma taxa de cruzamento definida *a priori*, e os descendentes gerados são submetidos à operação de mutação mediante uma taxa de mutação, com o objetivo de promover a diversidade. Assim, estes descendentes mutados (ou não) vão substituir os seus progenitores na população. Este processo dá origem a uma nova população, designada por nova geração (Pirlot, 1996; Talbi, 2009).

O algoritmo termina após ser atingido um determinado critério de paragem, podendo este critério ser referente a um número de gerações, ou referente à não verificação de melhoria na aptidão dos indivíduos criados. Cada indivíduo é avaliado pelo seu enquadramento ou aptidão, podendo este valor corresponder somente ao valor da função objetivo tanto num problema de maximização como de minimização (Madureira, 2003; Pirlot, 1996).

Os principais operadores usados nos Algoritmos Genéticos são a seleção, o cruzamento e a mutação (Gonzalez, 2007; Madureira, 2003; Pirlot, 1996; Talbi, 2009). O operador de seleção apresenta um papel fundamental nos Algoritmos Genéticos, dado que através dele são selecionados os indivíduos para reprodução em cada população. A forma de seleção mais evidente parece ser o elitismo onde, em analogia com o mundo real, os indivíduos mais fortes prevalecem sobre os mais fracos. Assim, poderiam ser descartados os indivíduos mais fracos, escolhendo-se os mais aptos para a população seguinte, e para reprodução. No entanto, no caso da resolução de problemas de otimização o mesmo não acontece, pois podem ser descartados indivíduos necessários para se chegar ao melhor indivíduo possível, através de operações de cruzamento e mutação. No entanto, poderá ser importante usar algum elitismo de forma a conservar os melhores indivíduos entre novas populações. Assim, o elitismo pode ser usado em conjunto com outros métodos de seleção (Luke, 2012).

Um dos primeiros métodos de seleção de indivíduos, referido na literatura, foi o método da roleta (*fitness proportionate selection*, ou *roulette wheel selection*, em inglês) (Bäck, 1994; Glover e Kochenberger, 2003; Talbi, 2009). Neste algoritmo, os indivíduos são selecionados proporcionalmente à sua aptidão, isto é, se um individuo tiver uma maior

aptidão então poderá ser selecionado mais vezes. O contrário acontecerá com indivíduos menos aptos. Uma variante deste método é a seleção estocástica universal, proposta por James Baker (1987), onde os indivíduos são selecionados igualmente com base na sua aptidão mas de uma forma tendenciosa para garantir que os indivíduos mais aptos sejam selecionados pelo menos uma vez. Uma vantagem em relação à seleção pelo método da roleta é que o melhor indivíduo será selecionado de certeza, até mais do que uma vez, o que não era garantido no método da roleta pois poderia acontecer que o melhor individuo nunca fosse selecionado. No entanto, estes métodos apresentam um problema relacionado com os valores das aptidões dos indivíduos. Imagine-se que, mesmo com uma função de aptidões escolhida cuidadosamente, no final duma geração os valores de aptidão de todos os indivíduos variam entre 9.97 e 9.99, numa escala entre 0 e 10. O objetivo seria escolher aqueles com valor de aptidão igual a 9.99 mas, de acordo com os métodos anteriormente descritos, todos os indivíduos teriam uma probabilidade de seleção muito idêntica. O sistema teria então convergido para uma seleção aleatória. O problema seria resolvido através da utilização de uma escala mais sensível aos valores mais altos ou através do uso de um algoritmo de seleção não-paramétrico. Surgiu assim o método de seleção por torneio, para colmatar as limitações dos métodos anteriores (Glover e Kochenberger, 2003; Luke, 2012).

O método de seleção por torneio baseia-se em selecionar os melhores indivíduos de um conjunto *t* de indivíduos selecionados aleatoriamente da população e foi usado pela primeira vez por Anne Brindle (1981), que usou uma seleção por torneio binária (*t*=2). Desde então, este método tornou-se o mais popular e a principal técnica usada para seleção nos Algoritmos Genéticos, devido a várias razões. Primeiro, porque não é sensível a particularidades da função de aptidão. Segundo, porque é bastante simples, não requer pré-processamento, e funciona bem com algoritmos paralelos. Terceiro, porque é parametrizável através da definição do tamanho do torneio *t,* sendo possível especificar se a técnica é muito ou pouco seletiva (Luke, 2012).

Outro dos principais operadores de reprodução dos Algoritmos Genéticos é o operador de cruzamento, que, tal como já referido, tem como função criar novos indivíduos (descendentes) a partir de outros selecionados para o efeito (progenitores). Os descendentes criados possuem uma combinação de códigos genéticos dos seus progenitores, tal como ocorre na recombinação natural. Se os cromossomas forem apenas cadeias de genes, essas combinações são feitas através de um ou mais pontos de cruzamento nos cromossomas dos seus ascendentes, sendo misturadas as partes resultantes dos cromossomas para cada descendente. Se com uma codificação binária não

for possível detalhar o contexto de um problema, deverá proceder-se à utilização de um alfabeto diferente, sendo nesse caso necessário especificar um operador de cruzamento apropriado (Glover e Kochenberger, 2003; Gonzalez, 2007; Madureira, 2003; Pirlot, 1996; Talbi, 2009). Os operadores de cruzamento mais comuns são (Figura 3.2):

- Cruzamento de ponto-único, onde é gerado um determinado índice e são trocados todos os genes maiores do que esse índice;
- Cruzamento de duplo ponto, onde são gerados dois índices e são trocados todos os genes entre esses dois pontos;
- Cruzamento uniforme, onde são trocados os valores dos genes de determinados índices gerados aleatoriamente.

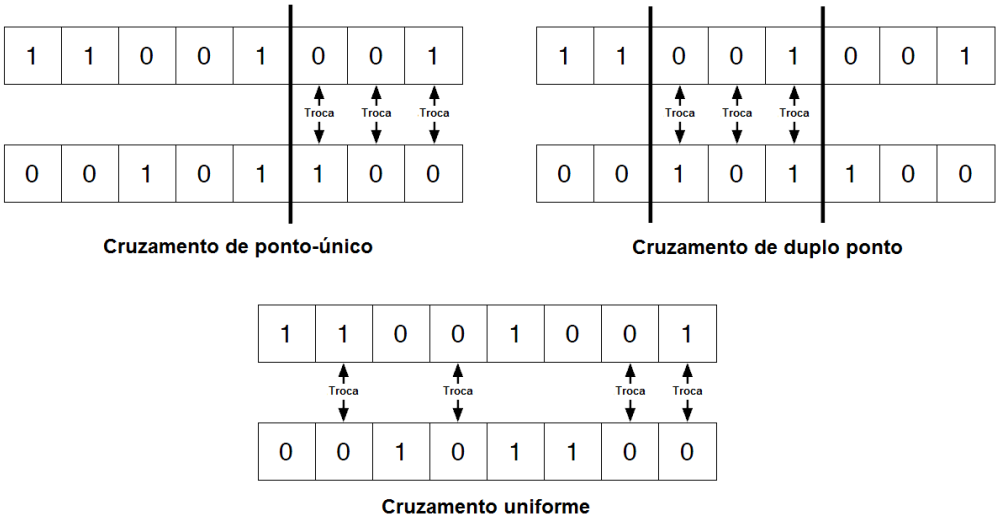

Figura 3.2 – Exemplos de cruzamento de cromossomas binários (adaptado de (Luke, 2012))

Finalmente, o operador de mutação consiste em trocar o valor de um ou mais genes num determinado cromossoma com uma probabilidade definida, resultante de uma taxa de mutação, de forma a tornar as populações mais diversificadas em termos genéticos. O que se pretende com este operador é gerar alguns pontos aleatórios no espaço de soluções para prevenir a convergência prematura para ótimos locais, e consequentemente poderem ser exploradas zonas mais alargadas. Os operadores de mutação mais conhecidos em problemas de escalonamento são (Madureira, 2003): o *swap*, que consiste em trocar dois genes selecionados aleatoriamente, e o *shift*, que permite deslocar um gene relativamente à sua posição, para a direita ou para a esquerda.

Os principais aspetos a ter em conta no desenvolvimento de Algoritmos Genéticos são a representação dos indivíduos, a inicialização da população, a função de aptidão

(função objetivo), as estratégias de seleção, de reprodução e de substituição, e o critério de paragem (Talbi, 2009).

Os Algoritmos Genéticos mostraram não ser muito adequados para problemas de escalonamento pois não são eficientes na obtenção de uma solução quase-ótima em tempo razoável, quando comparado com a Pesquisa Tabu e o *Simulated Annealing*, que operam com uma configuração única e não com uma população inteira de indivíduos (Ouelhadj e Petrovic, 2008). No entanto, os Algoritmos Genéticos têm sido aplicados na resolução de problemas de escalonamento em (Bouabda *et al.*, 2011; De Giovanni e Pezzella, 2010; Madureira *et al.*, 2009a; Mendes *et al.*, 2009; Pereira e Madureira, 2013; Pereira *et al.*, 2013a; Qing-dao-er-ji e Wang, 2012; Vallada e Ruiz, 2011; Vilcot e Billaut, 2008; Zhou *et al.*, 2009; Zhu *et al.*, 2011).

3.2.5. Otimização por Colónia de Formigas

A Meta-heurística Otimização por Colónia de Formigas (*Ant Colony Optimization*), baseia-se no comportamento cooperativo das formigas, ou seja, num comportamento que permite encontrar o menor caminho entre uma fonte de comida e a respetiva colónia (Blum e Roli, 2003; Glover e Kochenberger, 2003; Maniezzo *et al.*, 2004; Talbi, 2009). Este fenómeno ocorre porque, durante a sua trajetória, as formigas depositam no caminho uma substância chamada feromona e, ao escolherem um caminho, optam, com maior probabilidade, por aquele que possui a maior concentração de feromona, pois provavelmente será o mais curto, isto é, a trajetória que o maior número de formigas já realizou. Assim, os algoritmos de Colónia de Formigas são baseados na parametrização probabilística deste modelo, que propõe o uso de feromona para a definição de um caminho.

Numa colónia de formigas, quanto menor o caminho, maior será a concentração de feromona, que é uma substância olfativa e volátil. A feromona evapora-se ao longo do tempo e quanto maior for o percurso, maior será o seu impacto. Com o passar do tempo, as formigas seguem os caminhos com maior quantidade de feromona, e, no final, uma colónia de formigas é capaz de encontrar o caminho mais curto entre dois pontos (Glover e Kochenberger, 2003; Luke, 2012; Talbi, 2009).

A Figura 3.3 ilustra uma experiência feita por Goss *et al.* (1989) com uma colónia de formigas argentinas (*Iridomyrmex humilis*). A colónia tem acesso a uma fonte de comida através de dois caminhos e, durante a sua viagem, as formigas largam a feromona para marcar o caminho e guiar as outras formigas até à comida. Quanto maior for a quantidade

de feromona num caminho, maior será a probabilidade das formigas seguirem esse caminho. Além disso, tal como já referido, a feromona tem um efeito de evaporação ao longo do tempo. Isso significa que se as formigas demorarem muito tempo a percorrer um caminho, ou se o deixarem de percorrer, a feromona tende a evaporar-se naturalmente (Talbi, 2009).

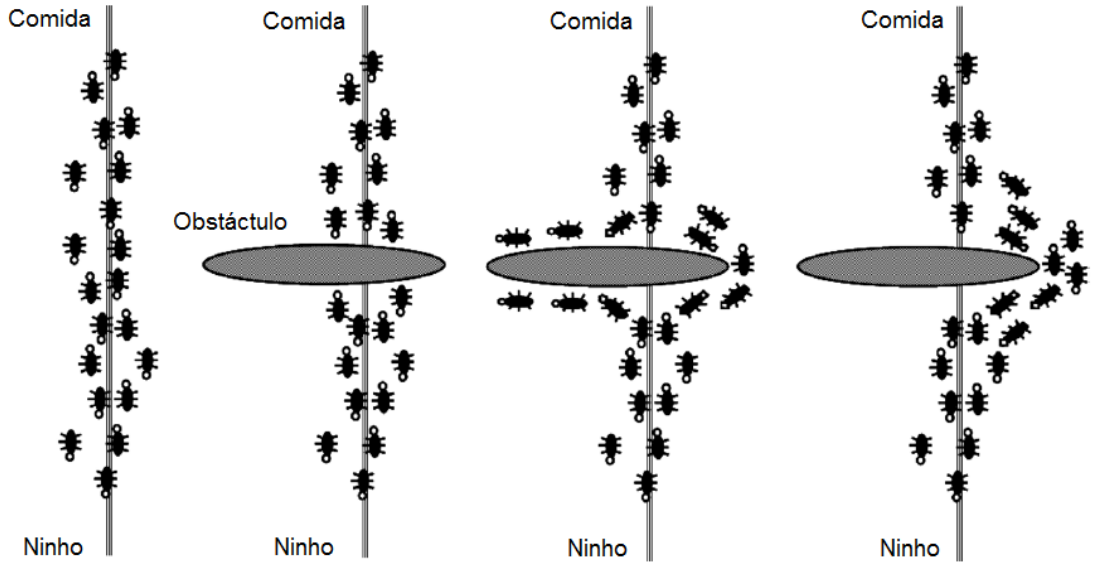

Figura 3.3 – Inspiração de uma colónia de formigas à procura do melhor caminho entre a comida e o ninho (adaptado de (Talbi, 2009))

Quando as formigas enfrentam um obstáculo existe igual probabilidade de escolherem o caminho da esquerda ou o da direita (Figura 3.3). Como o caminho da direita é mais curto do que o esquerdo, sendo mais rápido de atravessar, as formigas vão acabar por deixar um valor de feromona maior. Quantas mais formigas forem pelo caminho da direita, maior será a quantidade de feromona, havendo assim uma convergência para o caminho mais curto, pelo facto da feromona se evaporar no caminho esquerdo ao longo do tempo. Esta forma indireta de cooperação é denominada de estigmergia (*stigmergy* em inglês).

O primeiro sistema baseado nesta Meta-heurística foi introduzido por Marco Dorigo em 1991, com a designação de *Ant System* (Dorigo *et al.*, 1991).

Esta Meta-heurística (Algoritmo 3.5) começa pela inicialização da feromona com um valor único pré-definido. De seguida, para todas as iterações do algoritmo, cada formiga constrói o seu caminho/solução com base na feromona atual. Posteriormente, quando toda a população de formigas foi tratada, a feromona é atualizada com base nestas soluções previamente construídas. Assim, o algoritmo é composto principalmente por duas fases

iterativas: construção de soluções e atualização da feromona (Talbi, 2009). Na fase de construção, a formiga constrói incrementalmente a solução/caminho, adicionando componentes para a solução já existente. A escolha probabilística de um novo componente para o caminho/solução é feita com base na equação (3.3), onde α e β são parâmetros para ajustar a importância da informação heurística (η_r) e os valores de feromona (τ_r), respetivamente. $J(s_a[c_l])$ refere-se ao conjunto de componentes da solução $s_a[c_l]$ onde c_l é o último componente adicionado (Blum e Roli, 2003; Glover e Kochenberger, 2003; Maniezzo *et al.*, 2004; Talbi, 2009).

Algoritmo 3.5 – Otimização por Colónia de Formigas

```
    // Número de formigas, critério de paragem, e taxa de evaporação da feromona
    Entrada: NumFormigas, CritParagem, TxEvap
    Saída: MelhorSolucao // Melhor solução encontrada pelo algoritmo
1   NumIteracoes← 0;
2   MelhorSolucao ← ∅;
3   Populacao ← geraPopulacao(NumFormigas); // Gera uma população de formigas
4   inicializar(MatrizFeromona); // Inicializa a matriz de feromona;
5   Enquanto NumIteracoes++ < CritParagem Faz
6           Para cada Formiga_i ∈ Populacao Faz // Percorre todos as formigas da população
7                   // Constrói incrementalmente um caminho para a formiga (equação (3.3))
8                   construirCaminho(Formiga_i,MatrizFeromona);
9                   Se qualidade(Formiga_i) > qualidade(MelhorSolucao) Então
10                          // Atualiza a melhor solução encontrada
11                          MelhorSolucao ← Formiga_i;
12                  Fim
13          Fim
14          // Atualiza a feromona com base na equação (3.4)
15          atualizarFeromona(MatrizFeromona,TxEvap);
16  Fim
17  Devolver MelhorSolucao;
```

$$\mathbf{p}(C_r|s_a[c_l]) = \begin{cases} \text{Se } c_r \in J(s_a[c_l]): \dfrac{[\eta_r]^\alpha [\tau_r]^\beta}{\sum_{c_u \in J(s_a)[c_l]} [\eta_r]^\alpha [\tau_r]^\beta} \\ \text{Senão: } 0 \end{cases} \quad (3.3)$$

Na fase da atualização da feromona, quando todas as formigas já construíram uma solução, é aplicada uma regra definida pela equação (3.4), onde *F* é geralmente denominada por função de qualidade, definida pelo *fitness* das soluções (Blum e Roli, 2003; Glover e Kochenberger, 2003; Maniezzo *et al.*, 2004; Talbi, 2009).

$$\tau_j \leftarrow (1-\rho) * \tau_j + \sum_{a \in A} \Delta\tau_j^{s_a}$$
$$\forall T_j \in T \text{, onde } \Delta\tau_j^{s_a} = \begin{cases} \text{Se é um componente de } s_a\text{: } F(s_a) \\ \text{Senão: } 0 \end{cases} \quad (3.4)$$

Esta regra para a atualização da feromona visa atualizar o valor de feromona dos componentes que encontraram as melhores soluções. De salientar que a primeira parte da equação ($(1-\rho) * \tau_j$) indica a evaporação da feromona, de modo a que caminhos maus se degradem ao longo do tempo, com base numa taxa de evaporação ρ. O incremento de feromona é feito apenas nos caminhos percorridos pelas formigas. O critério de paragem pode ser com base num número máximo de iterações, ou então quando ocorre a estagnação da colónia, fenómeno que ocorre quando todas as formigas percorrem o mesmo percurso.

A Otimização por Colónia de Formigas tem sido aplicada a problemas de otimização combinatória e tem alcançado sucesso na resolução de diversos problemas diferentes (Dorigo e Stützle, 2003). Esta técnica tem sido aplicada na resolução de problemas de escalonamento, ver e.g. (Pereira e Madureira, 2013; Pereira *et al.*, 2013a; Tavares Neto e Godinho Filho, 2011; Xiang e Lee, 2008; Xing *et al.*, 2010; Xu *et al.*, 2012; Yagmahan e Yenisey, 2010; Yoshikawa e Terai, 2006; Zhang *et al.*, 2012).

3.2.6. *Particle Swarm Optimization*

O *Particle Swarm Optimization* é uma técnica estocástica baseada em populações, inspirada na *Swarm Intelligence*, desenvolvida por James Kennedy e Russell Eberhart (1995), que pretende simular um sistema social simplificado. A ideia base desta Meta-heurística passa por imitar o comportamento social que os bandos de pássaros ou cardumes assumem nos seus movimentos coordenados e sincronizados, com o objetivo de descobrir comida ou como mecanismo de autodefesa. Nestas sociedades, os comportamentos coordenados emergem sem nenhum controlo central (Talbi, 2009).

Ao contrário de outras técnicas baseadas em populações, esta não produz novas populações. Em vez disso mantém a mesma população estática criada inicialmente e vai ajustando os seus indivíduos de forma a descobrir novas soluções no espaço de pesquisa. Além disso, opera quase exclusivamente num espaço de soluções multidimensional e de valores reais. Por isso é que, em vez de populações de indivíduos, as soluções candidatas

são referidas como enxame de partículas (Luke, 2012). Nesta Meta-heurística, o processo de otimização tira vantagens do processo de cooperação entre as várias partículas, e o sucesso de algumas partículas influencia o comportamento dos seus pares. Assim, existem *N* partículas a pesquisar um espaço de soluções com *D* dimensões. Cada uma dessas partículas é uma solução candidata e tem a sua própria posição e velocidade no espaço, tal como ilustrado na Figura 3.4, ajustando a sua direção com base na sua própria experiência e na experiência das restantes partículas (Talbi, 2009).

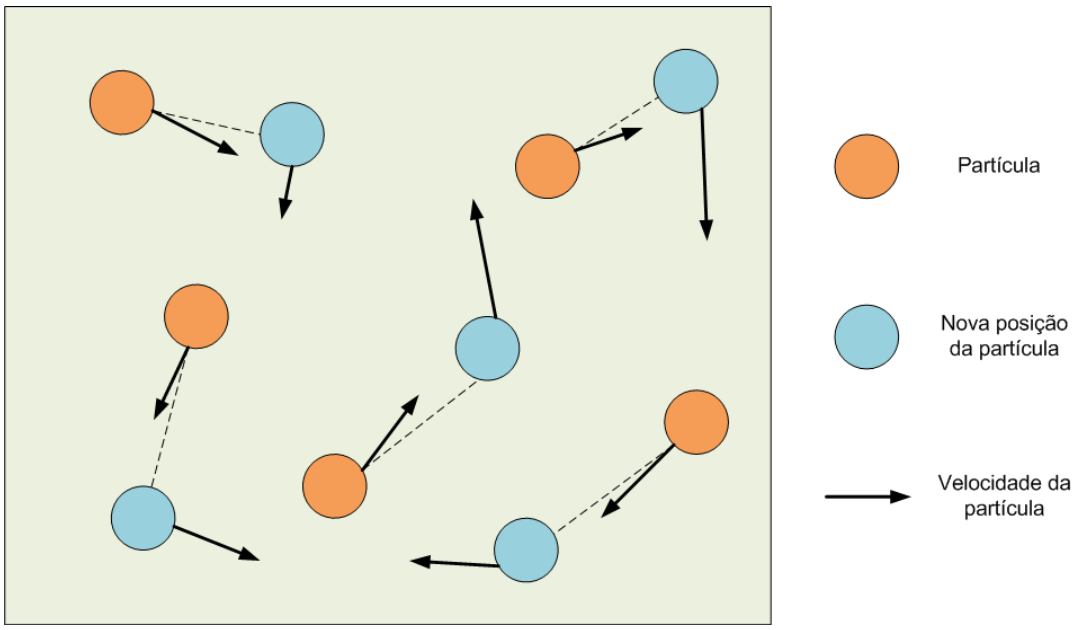

Figura 3.4 – Ilustração de um conjunto de partículas no *Particle Swarm Optimization* (adaptada de (Talbi, 2009))

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 cognitivo, o fator social, e a velocidade máxima (Kennedy e Eberhart, 1995). O fator cognitivo (C1) determina a atração da partícula com a sua melhor posição encontrada. O fator social (C2) determina a atração (convergência) das partículas para a melhor solução descoberta por um elemento do grupo. O fator de velocidade máxima delimita o movimento, uma vez que esse é direcional e determinado.

O *Particle Swarm Optimization* (Algoritmo 3.6) começa pela inicialização (aleatória ou não) da posição atual e velocidade de todas as partículas. De seguida, enquanto o critério de paragem não é atingido (normalmente número de iterações), em cada iteração, é calculado o valor de aptidão (*fitness*) para cada partícula, sendo atualizado o melhor valor

local de cada uma (*pBest*) e o melhor valor global (*gBest*). Depois de todas as partículas estarem tratadas, são calculadas as novas velocidades e posições de todas as partículas, como se descreve de seguida. 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 (3.5).

$$X_{id} = X_{id} + V_{id} \tag{3.5}$$

Algoritmo 3.6 – *Particle Swarm Optimization*

```
    Entrada: NumParticulas, CritParagem // Número de partículas e critério de paragem
    Saída: gBest // Melhor solução encontrada pelo algoritmo
1   NumIteracoes← 0;
2   Populacao ← geraPopulacao(NumParticulas); // Gera uma população de partículas
3   pBest ←Array(NumParticulas); // Vetor para guardar as melhores soluções das partículas
4   gBest ← ∅; // Melhor solução global
5   Enquanto NumIteracoes++ < CritParagem Faz
6           // Percorre todas as partículas calculando pBest e gBest
7           Para cada Particula_i ∈ Populacao Faz
8                   Se qualidade(Particula_i) > qualidade(pBest_i) Então
9                           // Atualiza a melhor solução local da partícula (pBest)
10                          pBest_i ← Particula_i;
11                  Fim
12                  Se qualidade(Particula_i) > qualidade(gBest) Então
13                          // Atualiza a melhor solução encontrada (gBest)
14                          gBest ← Particula_i;
15                  Fim
16          Fim
17          Para cada Particula_i ∈ Populacao Faz // Percorre todas as partículas
18                  // Atualiza a velocidade de acordo com a equação (3.6)
19                  atualizarVelocidade(Particula_i);
20                  // Atualiza a posição de acordo com a equação (3.5)
21                  atualizarPosicao(Particula_i);
22          Fim
23  Fim
24  Devolver gBest;
```

Além da sua posição no espaço de soluções, cada partícula tem associada, como já referido, a sua própria velocidade $V_i = \{V_{i1}, V_{i2}, \dots, V_{id}\}$. A velocidade de cada partícula consiste num vetor que representa a direção e a aceleração que a partícula possui em cada instante temporal, ao longo das iterações, e define a direção e a distância que as partículas devem percorrer no espaço de soluções. Assim, a velocidade de cada partícula é atualizada

com base na equação (3.6), onde R1 e R2 representam dois valores aleatórios no intervalo [0;1] e W representa a inércia associada à velocidade.

$$V_{id} = W \times V_{id} + C1 \times R1 \times (pBest_{id} - X_{id}) + C2 \times R2 \times (gBest_d - X_{id}) \quad (3.6)$$

Esta fórmula reflete um aspeto fundamental da sociabilidade humana onde a tendência sociopsicológica dos indivíduos emula os sucessos de outros indivíduos (Talbi, 2009). Tal como já referido, a velocidade V_{id} é limitada por um valor máximo que faz com que varie no intervalo $[-V_{max};+V_{max}]$ de modo a dar algum controlo à aleatoriedade associada.

A inércia W é usada 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; Talbi, 2009).

Esta Meta-heurística foi recentemente aplicada a problemas de escalonamento, ver e.g. (Chou, 2012; Liao *et al.*, 2012; Liu *et al.*, 2010; Moslehi e Mahnam, 2011; Pereira e Madureira, 2013; Pereira *et al.*, 2013a; Pongchairerks e Kachitvichyanukul, 2009; Yen e Ivers, 2009; Zelenka, 2011; Zelenka e Kasanicky, 2011).

3.2.7. Colónia de Abelhas Artificiais

Os algoritmos de otimização baseados nos comportamentos das abelhas são algoritmos estocásticos pertencentes à *Swarm Intelligence*. Os primeiros algoritmos surgiram em 2005 por Derviş Karaboğa (Karaboga, 2005) e Duc Truong Pham (Pham *et al.*, 2005), independentemente. Estes algoritmos são inspirados no comportamento duma colónia de abelhas à procura de comida, o que pode ser usado como modelo para comportamentos inteligentes e coletivos (Talbi, 2009).

Os enxames de abelhas vivem em colónias sociais, cujo ninho se designa por colmeia, e são conhecidas pela produção do mel e pela adaptação constante às mudanças no ambiente duma forma coletivamente inteligente. As abelhas têm memória fotográfica, sistemas de navegação próprios, usam o néctar como fonte de energia e consomem o pólen

como proteína para a criação das suas proles (Karaboga e Akay, 2009; Talbi, 2009). Geralmente, uma colónia de abelhas contém um reprodutor feminino (rainha), alguns milhares de machos (zangões), e muitos milhares de fêmeas estéreis (trabalhadoras). Depois de acasalar com vários zangões, a rainha gera muitas jovens abelhas dando início a uma nova geração (Karaboga e Akay, 2009; Talbi, 2009). A rainha tem uma esperança média de vida entre 3 a 5 anos, é a mãe de todos os membros da colónia, e a sua tarefa principal é o acasalamento com os zangões, numa operação reprodutiva que se designa por voo de acasalamento. Os ovos fertilizados tornam-se fêmeas (trabalhadoras e futura rainha) e os ovos não fertilizados tornam-se machos (zangões).

Os zangões são os machos na colmeia, têm uma esperança média de vida na ordem dos 90 dias e não vivem mais do que 6 meses. A sua principal tarefa é acasalarem com a rainha e morrem após o terem feito com sucesso.

As abelhas trabalhadoras são as fêmeas sem capacidade de reprodução, representam a maior parte das abelhas na colmeia, e vivem normalmente entre 4 a 9 meses. Na primeira metade da sua vida são responsáveis por muitas tarefas na colmeia, desde a manutenção até à defesa da mesma. Na segunda metade da sua vida, a principal tarefa destas abelhas passa pela procura de alimentos, sendo que inicialmente fazem voos curtos com o objetivo de aprenderem a localização da colmeia e a topologia do ambiente.

Em determinado momento, as abelhas trabalhadoras podem estar ou não empregadas (Karaboga, 2005). As abelhas “empregadas” estão associadas a uma determinada fonte de comida que estão constantemente a explorar e a extrair alimento. Estas abelhas transportam consigo informação acerca da respetiva fonte de comida, a sua distância e direção em relação à colmeia, a rentabilidade da fonte de comida e partilham essa informação na colmeia. As abelhas “desempregadas” estão continuamente à procura de uma fonte de comida para explorar e dividem-se em exploradoras, que procuram novas fontes de comida no ambiente circundante, e em espectadoras, que esperam na colmeia pelas abelhas empregadas para, através da informação partilhada, as ajudarem na exploração das fontes de alimento.

A tarefa de procura de alimentos é a tarefa mais importante de uma colónia de abelhas (Karaboga e Akay, 2009). Este processo começa com a procura de fontes de comida por parte de um grupo de abelhas exploradoras, com o objetivo de extrair néctar. O valor de uma fonte de comida depende de vários fatores tais como a proximidade da colmeia, a riqueza ou concentração de energia, e a facilidade de extração do alimento. Por uma questão de simplicidade, a rentabilidade de uma fonte de comida pode ser

representada por um único valor numérico que representa a sua qualidade (Karaboga, 2005). Depois de descarregar o néctar, a abelha exploradora que encontrou a fonte de comida torna-se "empregada" e efetua um movimento especial, uma espécie de dança, com o objetivo de partilhar informação acerca da fonte de comida para recrutar abelhas espectadoras para irem explorar essa fonte. Além da riqueza da fonte de comida, a informação partilhada envolve também a direção e a distância em relação à colmeia (Karaboga e Akay, 2009).

Tabela 3.3 – Analogia entre abelhas reais e abelhas artificiais (Talbi, 2009)

Colónia de abelhas naturais	Colónia de abelhas artificiais
Fonte de comida	Solução
Qualidade do néctar (*fitness*)	Função objetivo
Abelhas exploradoras	Prospeção de soluções
Abelhas espectadoras	Exploração do espaço de soluções

A partilha de informação entre as abelhas é a parte mais importante na formação do conhecimento coletivo. Assim, existem três tipos diferentes de danças que as abelhas "empregadas" podem fazer: *round dance*, *waggle dance*, e *tremble dance* (Karaboga, 2005; Karaboga e Akay, 2009). Se a distância da fonte de comida em relação à colmeia for inferior a 100 metros, as abelhas efetuam a *round dance*, senão efetuam a *waggle dance*. Enquanto a *round dance* não dá nenhuma informação acerca da direção da fonte de comida, a *waggle dance* dá informação acerca da direção em relação ao sol, da distância em relação à colmeia e da qualidade da fonte de comida. Por último, a *tremble dance* é usada quando a abelha deteta atrasos quando está a descarregar o seu néctar.

As abelhas espectadoras observam várias danças de diferentes abelhas e decidem a qual fonte de comida se querem empregar, existindo uma maior probabilidade de escolherem fontes mais rentáveis. Depois de escolherem e localizarem a fonte de comida, as abelhas espectadoras tornam-se em abelhas "empregadas", explorando essa fonte para extrair o seu néctar. Quando a fonte de comida se esgota, as abelhas "empregadas" tornam-se em exploradoras (Karaboga, 2005). Na Tabela 3.3 está descrita a analogia entre abelhas reais e artificiais.

Na Colónia de Abelhas Artificiais (Algoritmo 3.7), a posição da fonte de comida representa uma solução possível para a resolução do problema de otimização e a quantidade de néctar corresponde à qualidade (*fitness*) dessa solução. O número de abelhas "empregadas" é igual ao número de fontes de comida, cada uma representando um

local que é explorado pelas abelhas exploradoras. Cada ciclo do algoritmo é composto por três passos principais: mover as abelhas "empregadas" e espectadoras para as fontes de comida, calcular a quantidade de néctar, e determinar as abelhas exploradoras para as direcionar até outras fontes de comida (soluções) (Karaboga e Basturk, 2007).

No início, a população de *Sn* abelhas é inicializada de forma aleatória, sendo que *Sn* representa o tamanho da população e a cada abelha "empregada" é atribuída uma fonte de comida (solução).

Algoritmo 3.7 – Colónia de Abelhas Artificiais

```
    Entrada: NumAbelhas, CritParagem // Número de abelhas e critério de paragem
    Saída: MelhorSolucao // Melhor solução encontrada pelo algoritmo
1   NumIteracoes← 0;
2   Populacao ← geraPopulacao(NumAbelhas); // Gera uma população de abelhas
3   MelhorSolucao ← ∅;
4   Enquanto NumIteracoes++ < CritParagem Faz
5           Populacao.abelhasEmpregadas(); // Fase das abelhas "empregadas"
6           // Calcular probabilidades para as abelhas espectadoras com base na equação (3.7)
7           Populacao.calcProbsAbelhasEspectadoras();
8           Populacao.abelhasEspectadoras(); // Fase das abelhas espectadoras
9           Populacao.abelhasExploradoras(); // Fase das abelhas exploradoras
10          // Percorre todas as abelhas atualizando a melhor solução
11          Para cada Abelha_i ∈ Populacao Faz
12                  Se qualidade(Abelha_i) > qualidade(MelhorSolucao) Então
13                          MelhorSolucao ← Abelha_i;
14                  Fim
15          Fim
16  Fim
17  Devolver MelhorSolucao;
```

As abelhas espectadoras são colocadas nas fontes de comida de acordo com uma probabilidade que é diretamente proporcional à quantidade de néctar, isto é, à qualidade da solução. Na equação (3.7) está definida a fórmula de cálculo dessa probabilidade, onde $f(S)$ representa a qualidade da solução.

$$P = \frac{f(S)}{\sum_{i=1}^{Sn} f(S_i)} \tag{3.7}$$

Depois de selecionarem a fonte de comida, as abelhas espectadoras determinam um vizinho para explorar, de forma a ser realizada uma seleção "gananciosa" à volta da fonte de comida. Uma das formas de identificar um vizinho pode ser através da equação (3.8), mas podem ser usados outros métodos de cálculo de vizinhança. Nesta equação, S é a fonte de

comida, S' é o vizinho, θ é um valor aleatório no intervalo [-1;1], e S_k é outra fonte de comida selecionada ao acaso. A ideia é explorar o espaço de soluções entre S e S_k.

$$S' = S + \theta(S - S_k) \tag{3.8}$$

O algoritmo tem um parâmetro de controlo muito importante, uma espécie de "limite" (Karaboga, 2005). Se a solução que representa a fonte de comida não for melhorada num número predeterminado de tentativas significa que a fonte de comida se esgotou, e nesse caso a fonte de comida é abandonada e a abelha "empregada" torna-se em exploradora. O "limite" é então o número de tentativas para se libertar uma fonte de comida.

As abelhas exploradoras não têm qualquer orientação na procura de novas fontes de comida, por isso são caracterizadas por, normalmente, serem rápidas a encontrar fontes de comida, mas de baixa qualidade. No entanto, as abelhas exploradoras podem, ocasionalmente, descobrir fontes ricas de comida.

A Colónia de Abelhas Artificiais foi aplicada na resolução de problemas de escalonamento, ver e.g. (Banharnsakun *et al.*, 2012; Chong *et al.*, 2006; Pan *et al.*, 2011; Pansuwan *et al.*, 2010; Tasgetiren *et al.*, 2011; Xiao e Chen, 2011; Zhang *et al.*, 2012; Zhou *et al.*, 2012).

3.2.8. Outras Meta-heurísticas

A Pesquisa Local Aleatorizada (Luke, 2012; Madureira, 2003) é uma extensão da Pesquisa Local, e consiste em executar *N* vezes o algoritmo de Pesquisa Local, a partir de soluções iniciais geradas aleatoriamente. O objetivo é ultrapassar as limitações da Pesquisa Local, evitando alguns ótimos locais.

A Meta-heurística determinística e evolutiva *Scatter Search* teve origem num artigo de Fred Glover (1977) e usa recombinação de soluções selecionadas a partir de um conjunto de referência para construir outras soluções e dar origem à evolução da população (Glover e Kochenberger, 2003; Talbi, 2009). Começa por gerar uma população inicial que satisfaça um critério de diversidade e qualidade. O conjunto de referência é construído através da seleção de boas soluções presentes na população e as soluções selecionadas são combinadas entre si. A população é então atualizada de forma a ter tanto soluções de

grande qualidade como soluções diversificadas. O processo iterativo termina quando o critério de paragem for atingido.

O GRASP (*Greedy Random Adaptive Search Procedure*) consiste na repetição do algoritmo de Pesquisa Local a partir de soluções iniciais diferentes e é composto por duas fases: uma construtiva e outra de melhoramentos (Feo e Resende, 1989). Começa-se por construir uma solução inicial, utilizando um algoritmo aleatório "guloso" (*greedy*), depois aplica-se um algoritmo de Pesquisa Local para pesquisar a vizinhança dessa solução, guardando-se a melhor solução obtida. Estes dois passos são repetidos até que os critérios de paragem, normalmente o número máximo de iterações, serem atingidos (Glover e Kochenberger, 2003; Luke, 2012; Talbi, 2009).

Os Algoritmos Meméticos (Moscato, 1989) são uma variante dos Algoritmos Genéticos e consistem no refinamento dos indivíduos antes destes se submeterem às operações de recombinação e mutação (Glover e Kochenberger, 2003).

A Evolução Diferencial foi desenvolvida por Rainer Storn e Kenneth Price e a ideia base consiste em usar diferenças nos vetores para perturbar a população de vetores (Storn e Price, 1997). É um das melhores abordagens para a resolução de problemas de otimização contínua e usa um operador de mutação para procurar boas soluções. Se a população estiver muito espalhada no espaço de soluções, o operador de mutação fará mais mudanças. Caso contrário, se a população estiver condensada numa determinada região, as mutações serão reduzidas (Luke, 2012; Talbi, 2009).

Os Sistemas Imunitários Artificiais (Dasgupta, 1998) são sistemas adaptativos inspirados no Sistema Imunológico, o qual é altamente robusto, adaptativo, inerentemente paralelo e auto-organizado. Além disso, tem capacidades de memória e aprendizagem poderosas (Talbi, 2009). Um problema com solução desconhecida é tratado como antigénio enquanto potenciais soluções são modeladas como anticorpos.

A *Harmony Search* (Geem, 2000) baseia-se na analogia do processo de improvisação de música que ocorre quando um músico procura o melhor estado de harmonia nas melodias, como durante uma improvisação de jazz. O procedimento de otimização consiste em quatro passos: inicialização do problema de otimização e dos parâmetros do algoritmo, inicialização da memória *Harmony*, improvisação de uma nova harmonia a partir da procura de *Harmony*, e atualização da procura de *Harmony.* No final repetem-se os últimos dois passos até ser atingido o critério de paragem.

3.3. O problema da afinação de parâmetros

Qualquer Meta-heurística tem parâmetros que necessitam de ser definidos, os quais são, na maioria das vezes, afinados manualmente em procedimentos de tentativa-erro guiados por regras empíricas. A definição dos valores para os parâmetros de Meta-heurísticas e outros algoritmos é uma tarefa entediante e tem bastante impacto no seu desempenho, o que pode levar a um interesse considerável em vários mecanismos que podem tentar ajustar automaticamente os parâmetros para um dado problema (Birattari, 2009; Hutter *et al.*, 2006a; Smith, 2008). Além disso, Smit e Eiben (2010) relataram que alguns parâmetros são mais relevantes do que outros, uma vez que diferentes valores para esses parâmetros afetam mais o desempenho do que diferentes valores para outros parâmetros, e assim é necessário mais cuidado na definição dos seus valores.

Em geral, a afinação de parâmetros pode permitir uma maior flexibilidade e robustez mas requer uma inicialização cuidada, uma vez que a parametrização apresenta uma grande influência na eficiência e eficácia do processo de pesquisa (Smith, 2008). A definição *a priori* dos parâmetros não é óbvia pois estes dependem do problema e das instâncias a tratar, assim como do tempo que o utilizador dispõe para resolver o problema. Desse modo, a escolha dos valores dos parâmetros é considerada uma tarefa difícil. Além disso, não é possível obter valores ótimos para os parâmetros das Meta-heurísticas (Bartz-Beielstein *et al.*, 2004; Smith, 2008; Talbi, 2009). Existem também teoremas teóricos que provam a impossibilidade de obter uma boa definição de parâmetros genérica para todos os tipos de problema (e.g., teoremas *No Free Lunch* (Wolpert e Macready, 1997)).

Um dos maiores desafios que ainda persiste na área da computação evolucionária passa precisamente por encontrar os parâmetros apropriados para os algoritmos evolucionários (seleção, mutação, cruzamento, tamanho da população, etc.), sendo unânime que bons valores para os parâmetros são essenciais para um bom desempenho dos algoritmos evolucionários (Eiben e Smit, 2012; Nannen e Eiben, 2006). Na prática, os valores dos parâmetros são selecionados por convenções (a taxa de mutação deve ser baixa), escolhas *ad hoc* (porque não usar cruzamento uniforme?), e comparações experimentais em escalas limitadas (testar combinações em três taxas de cruzamento e três taxas de mutação diferentes). No entanto, começam a aparecer bons métodos de afinação nesta área que permitem aos utilizadores impulsionar o desempenho com custos moderados através da descoberta de bons valores de parâmetros adaptados ao problema corrente (Eiben e Smit, 2012).

Perante isto, várias abordagens têm sido propostas para a resolução do problema de parametrização das Meta-heurísticas. Na literatura é possível encontrar abordagens que tentam encontrar a melhor configuração de parâmetros para um dado problema (Adenso-Diaz e Laguna, 2006; Birattari *et al.*, 2002; Hutter *et al.*, 2006a). Todas estas técnicas pretendem encontrar uma definição de parâmetros para otimizar uma função objetivo que cubra todas as instâncias de um problema. Se as instâncias forem homogéneas então esta abordagem comporta-se bem. No entanto, se as instâncias vêm de distribuições heterogéneas ou mesmo de áreas de aplicação não relacionadas, então a melhor configuração de parâmetros pode variar de instância para instância.

Um problema relacionado com a afinação de Meta-heurísticas é o problema de seleção de algoritmos (Rice, 1975), que pode ser dividido em, pelo menos, duas perspetivas:

- Selecionar técnicas de resolução de algoritmos a partir de um conjunto de técnicas disponíveis, onde a ideia passa por gerir um conjunto de algoritmos e usar técnicas de aprendizagem de modo a construir modelos de previsão para escolher as técnicas mais adequadas, de acordo com um conjunto de características da instância do problema a resolver. Esta abordagem está relacionada com a noção de Híper-heurísticas e tem sido bem-sucedida na resolução de vários tipos de problemas (Gagliolo e Schmidhuber, 2010; Hutter *et al.*, 2006a; Xu *et al.*, 2008);
- Afinar um algoritmo para resolver um conjunto de instâncias de um problema, que passa por determinar os valores apropriados para os parâmetros que podem ser categóricos ou numéricos. Na prática, tal como acontece na computação evolucionária, a afinação desses parâmetros é realizada manualmente através da aplicação de algumas regras empíricas e convenções. O uso de ferramentas automatizadas para encontrar definições de parâmetros que otimizem o desempenho tem o potencial para libertar os programadores da tarefa entediante de afinar algoritmos manualmente. De notar que a tarefa de construir um algoritmo através da combinação de vários componentes pode ser um caso especial de configuração de algoritmos (Hutter *et al.*, 2007; Smit e Eiben, 2009).

Existem duas abordagens diferentes para a afinação de parâmetros das Meta-heurísticas (Talbi, 2009): a inicialização de parâmetros *offline* e a estratégia de afinação *online* de parâmetros (ver Figura 3.5). Na parametrização *offline*, os valores dos vários parâmetros são definidos antes da execução da Meta-heurística, enquanto na abordagem

online os parâmetros são controlados e atualizados de forma dinâmica ou adaptativa durante a execução da técnica. Estas duas abordagens são também conhecidas por afinação e controlo de parâmetros, respetivamente (Eiben e Smit, 2012; Hamadi *et al.*, 2012; Nannen *et al.*, 2008; Smith, 2008).

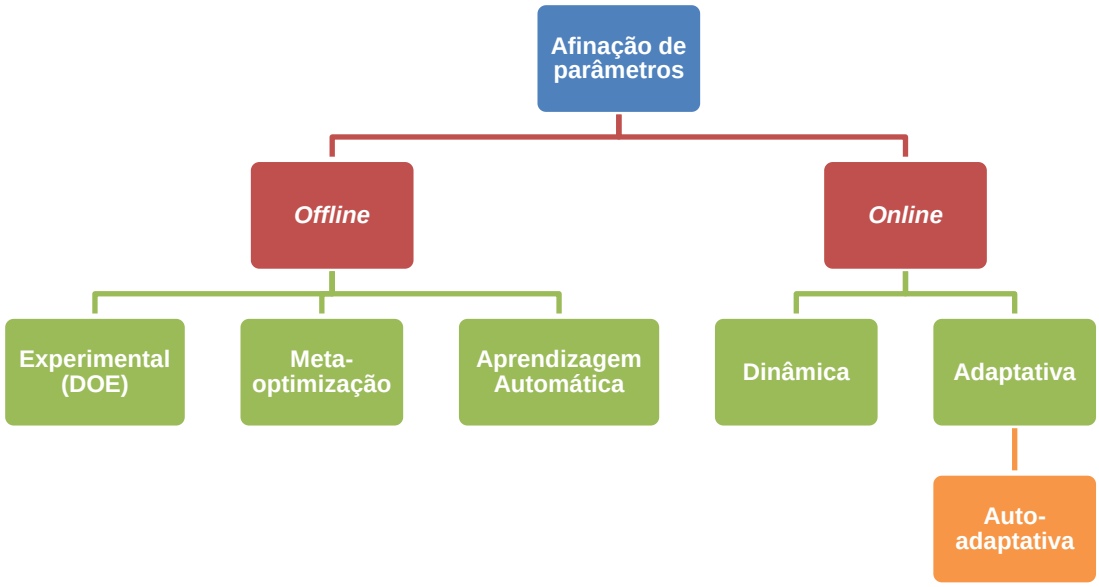

Figura 3.5 – Estratégias de definição de parâmetros de Meta-heurísticas (adaptado de (Talbi, 2009))

3.3.1. Parametrização *offline*

Tal como já foi referido, as Meta-heurísticas apresentam uma grande limitação pois necessitam que os seus parâmetros sejam afinados, o que não é fácil de realizar de forma completa. Os parâmetros não consistem apenas em valores numéricos mas também podem envolver o uso de componentes do processo de pesquisa (Talbi, 2009).

O objetivo da parametrização *offline* é obter valores que possam ser úteis para a resolução de um grande número de instâncias, o que requer um grande número de avaliações experimentais (Hamadi *et al.*, 2012). Normalmente, quem usa as Meta-heurísticas afina um parâmetro de cada vez, e o seu valor ótimo é determinado de forma empírica. Desta forma não é estudada a interação entre os parâmetros. Esta estratégia de otimização sequencial não garante a obtenção da configuração ótima mesmo se usar uma configuração de otimização exata.

Para superar este problema é muitas vezes usada uma abordagem experimental (*Design of Experiments* – DOE) (Bartz-Beielstein, 2006; Box *et al.*, 2005; Coy *et al.*, 2001).

Antes de ser possível usar uma abordagem experimental é necessário ter em conta os fatores que representam a variação de parâmetros na experiência e os níveis que representam os diferentes valores para os parâmetros (que podem ser quantitativos ou qualitativos), o que faz com que esta seja uma tarefa muito morosa (Smith, 2008). Johnson (2002) discutiu vários aspetos de especificação e análise de experiências com algoritmos de otimização estocásticos.

A grande desvantagem do uso de uma abordagem experimental consiste no elevado custo computacional quando existe um grande número de parâmetros e quando os domínios dos respetivos valores é igualmente elevado, uma vez que é necessário efetuar um grande número de experiências (Schaffer *et al.*, 1989). No entanto, é possível realizar poucas experiências com recurso ao hipercubo latino (McKay *et al.*, 1979) ou a modelos fracionários (Montgomery, 2008).

Outras desvantagens das abordagens DOE incluem (Smith, 2008):

- Os parâmetros não são independentes e tentar todas as combinações diferentes sistematicamente é praticamente impossível;
- Uma vez que as Meta-heurísticas são algoritmos estocásticos, é necessário realizar múltiplas execuções para cada combinação de modo a estabelecer corretamente diferenças significativas;
- O processo de afinação de parâmetros é moroso, mesmo se os parâmetros forem otimizados um por um, desprezando as suas interações;
- Para um dado problema, os valores dos parâmetros não são necessariamente ótimos, mesmo se o esforço feito para a sua definição for significativo.

Apesar de todas as desvantagens do DOE, é possível encontrar casos de sucesso. Santner *et al.* (2003) descreveu alguns métodos estatísticos modernos para desenhar e analisar experiências computacionais determinísticas, conhecidas como especificação e análise de experiências computacionais (*Design and Analysis of Computer Experiments* – DACE), e aplicou-os com sucesso para reduzir o custo computacional de problemas de otimização. Em (Bartz-Beielstein *et al.*, 2004) os autores usam uma metodologia sequencial flexível para análise de experiências de algoritmos de otimização, que emprega métodos computacionais estatísticos para investigar as interações entre problemas de otimização, algoritmos, e ambientes. Paralelamente, Bartz-Beielstein e Markon (2004) propuseram uma abordagem que combina métodos de DOE, DACE, e análise de regressão baseada em árvore para uma afinação de parâmetros mais eficiente e eficaz, que pode também ser

aplicada à análise de influência de diferentes parâmetros ou comparar o desempenho de vários algoritmos diferentes. No final dos anos 1990, Harik e Lobo (1999) deram uma contribuição à computação evolucionária ao apresentarem um algoritmo genético sem parâmetros resultante de um estudo dos melhores valores para a maioria das instâncias, de modo a não cair em casos extremos de seleção muito altos e muitos baixos. O objetivo seria ultrapassar as limitações resultantes do estudo de De Jong (1975), que propôs um conjunto de parâmetros padrão. Assim, o novo algoritmo genético alivia o utilizador da tarefa de definição de parâmetros e vai de encontro à proposta original de Holland (1975) que visualizou os algoritmos genéticos como métodos eficientes, fáceis de usar, e aplicáveis a uma grande variedade de problemas. No entanto, o algoritmo genético sem parâmetros é mais lento do que o normal.

Na afinação *offline* de parâmetros, a procura pelos melhores parâmetros para uma determinada Meta-heurística na resolução de um dado problema pode ser vista como um verdadeiro problema de otimização (Eiben e Smit, 2012; Hutter *et al.*, 2007). Por isso, uma abordagem de meta-otimização pode ser feita por uma qualquer (meta) heurística, levando a uma meta-Meta-heurística (ou meta-algoritmo). Esta abordagem pode ser considerada como um esquema híbrido, i.e., Meta-heurísticas híbridas (Blum e Roli, 2003; Talbi, 2009).

A meta-otimização é composta por dois níveis: o meta-nível e o nível base. No meta-nível as soluções representam os parâmetros a otimizar tais como o tamanho da lista tabu na Pesquisa Tabu, a taxa de arrefecimento no *Simulated Annealing*, as taxas de cruzamento e mutação de um Algoritmo Genético, etc. Neste nível, a função objetivo de uma solução corresponde à melhor solução encontrada (ou outro indicador de desempenho) pela Meta-heurística usada com os parâmetros especificados. Assim, para cada solução no meta-nível corresponde uma Meta-heurística independente no nível base.

Um exemplo de meta-otimização pode ser encontrado no trabalho de Hutter *et al.* (2007), que apresentaram uma abordagem baseada em Pesquisa Local para a configuração de algoritmos e provaram a sua convergência para a configuração ótima do algoritmo. Esta abordagem é muito versátil uma vez que pode ser usada para a minimização de tempo de execução em problemas de decisão ou para a maximização da qualidade das soluções em problemas de otimização. A validade desta abordagem foi demonstrada com sucesso num conjunto de estudos nos quais o objetivo era minimizar o tempo de execução de algoritmos (Birattari, 2009). O trabalho destes autores apresenta três contribuições principais:

- Definiram o primeiro algoritmo de Pesquisa Local Iterativa para o problema de configuração de algoritmos. Esta abordagem funciona em algoritmos

determinísticos e aleatórios e pode ser aplicada indiferentemente do cenário de afinação e o objetivo de otimização;

- Estudaram os efeitos da *over-confidence* e *over-tuning* que ocorrem quando um algoritmo é afinado com base num número finito de instâncias de treino (a abordagem dominante na literatura).
- Sugeriram uma versão estendida do algoritmo básico de modo a evitar *over-confidence* e *over-tuning*.

A meta-otimização é também bastante usada na área da computação evolucionária, maioritariamente através dos Meta-EA que funcionam como procedimentos de otimização de alto-nível para conduzir a pesquisa no espaço de soluções de parâmetros. Nestes métodos de pesquisa de alto-nível, todos os parâmetros são normalmente codificados num vetor e cada instanciação de uma instância desse vetor/indivíduo/algoritmo corresponde a um algoritmo evolucionário totalmente especificado. Consequentemente, o processo de afinação de parâmetros passa por encontrar o indivíduo ou algoritmo que produza o melhor desempenho (Yuan e Gallagher, 2007). Clune *et al.* (2005) concluíram que "*em situações nas quais o investimento humano requerido para definir os valores para os parâmetros é mais precioso do que o desempenho, então é preferível usar Meta-EA*".

Outras abordagens de meta-otimização aplicadas à resolução do problema de afinação de parâmetros de Meta-heurísticas podem ser encontradas em (Cook e Tauritz, 2010; Dréo, 2009; Goldman e Tauritz, 2011; Hansen, 2006; Rudolph *et al.*, 2009). Recentemente, vários autores têm proposto soluções para afinação de parâmetros com base em técnicas de Aprendizagem Automática, algumas delas descritas de seguida.

Nos casos em que as instâncias de um problema têm distribuições heterogéneas ou vêm de áreas de aplicação não relacionadas, a melhor configuração de parâmetros pode variar de instância para instância. Em (Hutter *et al.*, 2006a), os autores demonstraram que os modelos de aprendizagem automática podem fazer previsões surpreendentemente precisas das distribuições de tempo de execução de métodos de pesquisa aleatórios e incompletos, tais como as Meta-heurísticas, e como esses modelos podem ser usados para automaticamente ajustar os parâmetros para cada instância de um problema de modo a otimizar o desempenho, sem necessitar de intervenção humana. Trabalhos anteriores (Leyton-Brown *et al.*, 2006; Nudelman *et al.*, 2004) já tinham mostrado como é possível prever o tempo de execução de algoritmos de pesquisa em árvore determinísticos para problemas combinatórios usando técnicas de aprendizagem supervisionada. Os resultados obtidos indicam que a metodologia proposta por estes autores é capaz de prever tempos de

execução por instância e por parâmetro com uma precisão aceitável. Isto permite usar um modelo aprendido para automaticamente afinar os valores dos parâmetros para um algoritmo por instância, através de simplesmente utilizar a configuração de parâmetros que se prevê que produza o tempo de execução mais baixo. Uma afinação de parâmetros por instância é mais poderosa mas menos geral do que uma afinação com base numa distribuição, pois requer a existência de um conjunto de características discriminatórias das instâncias.

O conceito de *Racing* foi introduzido por Maron e Moore (1993; 1997), originalmente proposto com o objetivo de resolver eficientemente o problema de seleção de modelos em Aprendizagem Automática (Birattari, 2009). Quando aplicado à afinação de algoritmos, o *Racing* diminui o número de testes necessários para estimar a qualidade dos parâmetros e assim reduz o tempo total de execução do processo de afinação, sendo muito mais eficiente do que uma pesquisa exaustiva (Smit e Eiben, 2009; Yuan e Gallagher, 2007). Considere-se um conjunto de *M* combinações de parâmetros a serem testados em *N* instâncias de um problema. Através do uso do *Racing* não é necessário executar as combinações *M x N* vezes, sendo possível fazer um número reduzido de testes para cada combinação e realçar apenas aquelas que foram claramente boas (Smit e Eiben, 2009). A ideia básica por trás deste conceito é testar todas as combinações de parâmetros em paralelo e assim estas são forçadas a competir umas contra as outras, sendo que apenas as mais promissoras podem sobreviver para a próxima iteração. Como resultado, o custo computacional global é reduzido significativamente pois evita-se executar experiências desnecessárias (Yuan e Gallagher, 2007). O principal componente da técnica de *Racing* é o teste estatístico usado para julgar se um modelo é significativamente melhor ou pior do que outro, o que determina diretamente a eficiência e viabilidade da técnica (Yuan e Gallagher, 2007). Os algoritmos de *Racing* têm vindo a ser usados com bastante sucesso para a afinação de parâmetros, ver e.g. (Balaprakash *et al.*, 2007; Balaprakash *et al.*, 2009; Birattari, 2009; Birattari *et al.*, 2002; Socha, 2009; Yuan e Gallagher, 2004). Mais detalhes podem ser encontrados na secção 6.4.1.

Yuan e Gallagher (2007) mostraram como o *Racing* pode ser combinado com Meta-EA de modo a reduzir tanto o número de combinações de parâmetros como o número de testes e propuseram duas abordagens diferentes. Na primeira abordagem, os parâmetros numéricos representados por um vetor evoluem através de um Meta-EA e, a cada geração, é criado um conjunto de *N* novos vetores de acordo com uma distribuição Gaussiana centrada no melhor vetor atual. O *Racing* é então usado para determinar qual o vetor que teve a maior utilidade, e assim nem todos os *N* indivíduos precisam de ser avaliados o

mesmo número de vezes para determinar o melhor vetor. É expectável que muitos dos vetores sejam eliminados depois de um pequeno número de testes, economizando uma grande porção de tempo computacional. A segunda abordagem usa uma população de vetores de parâmetros contendo os parâmetros quantitativos (numéricos), que evoluem através da seleção, recombinação, e mutação. No entanto, a utilidade de um vetor com parâmetros numéricos é avaliada não apenas dentro de uma única instância, mas em todas as possíveis combinações de parâmetros qualitativos. A utilidade do vetor é igual ao desempenho da melhor combinação. O *Racing* é usado para determinar qual a combinação melhor sucedida usando o vetor de parâmetros numérico, e assim nem todas as combinações têm de ser avaliadas o mesmo número de vezes. Este processo economiza recursos computacionais, enquanto o espaço de procura de parâmetros quantitativos e qualitativos pode ainda ser explorado (Eiben e Smit, 2012; Yuan e Gallagher, 2007).

O método REVAC (*Relevance Estimation and Value Calibration*) foi introduzido por Nannen e Eiben (2006; 2007) para estimar a relevância e calibrar os valores dos parâmetros de um algoritmo evolucionário através da otimização de distribuições de probabilidade marginais sobre os parâmetros numéricos. Em (Nannen *et al.*, 2008) é demonstrado como o REVAC pode ser usado para indicar os custos e os benefícios de afinar cada parâmetro, algo que os Meta-EA não conseguem efetuar. O REVAC++ é a mais recente variante deste método, cujo nome foi motivado pelo facto do método conter dois módulos que visam melhorar o desempenho, sendo um deles o *Racing*. Esta versão do REVAC foi introduzida por Smit e Eiben (2009) onde também são mostrados os efeitos vantajosos do *Racing* quando usado com o REVAC. Os autores concluíram que usar metodologias de afinação de parâmetros é vantajoso uma vez que permite alcançar melhores desempenhos do que usar apenas convenções e regras empíricas. Para uma discussão de distribuições de probabilidade do REVAC, entropia, e relevância de parâmetros, ver (Smit e Eiben, 2010).

A Pesquisa Contínua surge como um novo conceito que consiste não só em resolver as novas instâncias do problema com base em conhecimento aprendido mas também melhorar continuamente o modelo de aprendizagem (Hamadi *et al.*, 2012). Neste conceito é possível encaixar o Raciocínio baseado em Casos (Aamodt e Plaza, 1994; Kolodner, 1993), uma técnica de Aprendizagem Automática que resolve problemas a partir de conhecimento e experiência adquirida de casos anteriores similares. As soluções ou estratégias de resolução de problemas que foram usadas na resolução de problemas anteriores (casos) são mantidas numa base de dados (base de casos) para poderem ser reutilizadas. Grolimund e Ganascia (1997) introduziram uma abordagem de Raciocínio baseado em Casos para realizar automaticamente o controlo do operador de seleção da Pesquisa Tabu.

Burke *et al.* (2003) propuseram uma Híper-heurística baseada em casos para resolver problemas de escalonamento de horários. Um horário é construído através da aplicação iterativa de um número de heurísticas, as quais podem ser selecionadas por um controlador de Raciocínio baseado em Casos. A ideia básica passa por manter uma base de casos de informação sobre as heurísticas mais bem-sucedidas para um grupo de problemas de escalonamento de horários conhecidos para prever a melhor heurística para um novo problema usando o conhecimento anterior. Pavón *et al.* (2009) apresentaram uma abordagem para afinação de parâmetros de algoritmos que usa Redes Bayesianas e Raciocínio baseado em Casos de modo a, automaticamente, selecionar a melhor configuração de parâmetros para uma dada instância de um problema, de acordo com as suas características e experiência anterior. Lin e Chen (2011) desenvolveram uma abordagem de Raciocínio baseado em Casos e Sistemas Imunitários Artificiais para aumentar a eficácia da classificação através da melhoria na afinação de parâmetros, da seleção de características e atribuição de peso das caraterísticas. Pereira e Madureira (2013) utilizaram o Raciocínio baseado em Casos para fazer uma parametrização autónoma de Meta-heurísticas, integrado num Sistema Multiagente para a resolução de problemas de escalonamento dinâmicos. Esta é uma das principais técnicas usadas no desenvolvimento deste trabalho, pelo que uma descrição mais detalhada pode ser encontrada na secção 6.4.2.

Outras abordagens de afinação automática de parâmetros incluem (Adenso-Diaz e Laguna, 2006; Coy *et al.*, 2001; Dobslaw, 2010; Oltean, 2005; Smith-Miles, 2008; Stoean *et al.*, 2009; Zennaki e Ech-Cherif, 2010).

3.3.2. Parametrização *online*

A desvantagem das abordagens *offline* é o alto custo computacional, especialmente se forem usadas para cada instância do problema. De facto, os valores ótimos para os parâmetros dependem do problema a resolver e até das diferentes instâncias (e.g., instâncias de maior dimensão podem necessitar de uma parametrização diferente daquelas usadas em instâncias mais pequenas). Assim, para aumentar a eficácia e a robustez das abordagens *offline,* estas devem ser aplicadas a todas as instâncias (ou grupo de instâncias) de um dado problema (Talbi, 2009).

Outra desvantagem das estratégias *offline* é que a eficácia da definição dos parâmetros pode ser alterada durante o processo de pesquisa, uma vez que, em determinados momentos da procura, diferentes valores ótimos são associados com um

determinado parâmetro. Além disso, alguns autores argumentam que deixar os parâmetros fixos durante a execução de um algoritmo parece ser inapropriado (Smith, 2008).

A ideia de algoritmos de pesquisa que conseguem adaptar automaticamente os seus parâmetros tem gerado um interesse considerável entre os investigadores (Smith, 2008). Entram assim em ação as abordagens *online* que monitorizam o progresso do processo de pesquisa e ajustam os valores dos parâmetros em tempo real.

As abordagens *online* podem ser divididas em dinâmicas e adaptativas (Bäck, 2001; Eiben e Smit, 2012; Hamadi *et al.*, 2012; Talbi, 2009). Nas abordagens dinâmicas, as alterações dos valores dos parâmetros são efetuadas de forma aleatória ou determinística, sem ter em conta o processo de pesquisa. Nas abordagens adaptativas os valores dos parâmetros mudam de acordo com o processo de pesquisa, através do uso de memória ou regras pré-estabelecidas. Uma subclasse, usada frequentemente na comunidade de computação evolucionária, é o conjunto das abordagens autoadaptativas que consistem em evoluir os parâmetros durante a pesquisa. Assim, os parâmetros são codificados e estão sujeitos a alterações, tal como as soluções para o problema (Smith, 2008).

O conceito de autoadaptação surgiu na área da computação evolucionária na década de 1970 e pode ser definido como uma propriedade dos sistemas naturais e artificiais que lhes permite controlar a forma como se adaptam às mudanças de ambiente (Smith, 2008).

Um algoritmo autoadaptativo deve exibir um controlo automático dos seus operadores ou parâmetros e, para isso, cada indivíduo codifica os seus próprios parâmetros. De modo a que a autoadaptação seja efetiva, deve existir uma variedade de conjuntos de parâmetros diferentes, de modo a dar alguma diversidade ao processo de evolução. Devem também existir ligações entre os parâmetros codificados e a respetiva mudança na codificação do problema bem como ligações entre as ações dos operadores e as subsequentes mudanças na qualidade das soluções (Smith, 2008).

Neste contexto, surgem várias técnicas de Aprendizagem Automática de modo a dotar os algoritmos com comportamentos autoadaptativos. A Pesquisa Reativa, proposta por Battiti et al. (2010; 2008) é caracterizada pela integração de técnicas de aprendizagem automática nas heurísticas de pesquisa, de modo a obter *feedback* durante o processo de pesquisa e modificar alguns parâmetros. Algumas abordagens deste tipo de pesquisa podem ser encontradas em (Hoos, 2002; Hutter *et al.*, 2006b). Hutter *et al.* (2006a) identificaram o uso de Aprendizagem por Reforço (Lagoudakis e Littman, 2001) e Aprendizagem Ativa (Epstein *et al.*, 2006).

O livro editado por Lobo *et al.* (2007) contém um conjunto de trabalhos interessantes na definição de parâmetros em computação evolucionária que foca principalmente a afinação *online*. Outras abordagens adaptativas aplicadas à resolução do problema de afinação de parâmetros de Meta-heurísticas podem ser encontradas em (Brunato e Battiti, 2008; Hartmann, 2002; Pan *et al.*, 2010; Peterson, 2011; Qin *et al.*, 2009; Stützle *et al.*, 2012; Yang *et al.*, 2008).

Tal como Bäck (2001) referiu, o princípio adjacente à autoadaptação de parâmetros é um dos métodos mais sofisticados para resolver o problema de afinação de algoritmos. As abordagens *online* podem ser mais poderosas do que as abordagens *offline*, mas são também menos gerais, uma vez que a sua implementação está intimamente ligada a um algoritmo específico (Hutter *et al.*, 2006a). Na opinião de De Jong (2007), para problemas de otimização estáticos, a pré-afinação de Meta-heurísticas irá continuar a ser mais adequada em relação à autoadaptação. Também Birattari (2009) referiu que as abordagens *online* de afinação de parâmetros são particularmente apelativas quando o objetivo é resolver um pequeno número de instâncias de grande dimensão e complexidade significativa. No entanto, contrastam com as abordagens *offline* que são mais adequadas para situações onde existe um grande número de instâncias para resolver.

3.4. Sumário

Neste capítulo foi explicado com algum detalhe o conceito de Meta-heurísticas, tendo sido efetuada uma descrição detalhada das técnicas utilizadas ao longo deste trabalho, nomeadamente a Pesquisa Tabu, os Algoritmos Genéticos, o *Simulated Annealing*, a Otimização por Colónia de Formigas, o *Particle Swarm Optimization*, e as Colónias de Abelhas Artificiais. Para cada uma delas foram identificados os parâmetros e também referidos alguns trabalhos aplicados ao problema de Escalonamento.

Além da descrição das várias técnicas, foi introduzido o problema da afinação de parâmetros de Meta-heurísticas, com bastante detalhe na parametrização *offline*, onde se destacam as abordagens experimentais, a meta-otimização e as abordagens de Aprendizagem Automática. Acerca das abordagens de parametrização *online*, foi dado particular ênfase às abordagens autoadaptativas.

Capítulo 4. Sistemas Multiagente

4.1. Introdução

Os Sistemas Multiagente têm sido estudados desde os anos 1980 e começaram a ganhar grande reconhecimento a partir dos anos 1990 (Bellifemine *et al.*, 2007; Reis, 2003; Wooldridge, 2002). Este rápido crescimento aconteceu em parte devido à crença por parte dos investigadores que os agentes constituem um paradigma de *software* apropriado para explorar os sistemas distribuídos, abertos e massivos, tal como a Internet (Reis, 2003; Wooldridge, 2002). Outra razão para este crescimento exponencial está relacionado com o facto destes se mostrarem adequados para a resolução de problemas cuja abordagem centralizada não demonstra ter capacidade de os resolver de forma satisfatória (Reis, 2003).

Os Sistemas Multiagente surgiram da Inteligência Artificial Distribuída, que interliga conceitos de Inteligência Artificial e de Computação Distribuída e tem evoluído e diversificado rapidamente desde a sua proposta em meados dos anos 1970 (Bellifemine *et al.*, 2007; Reis, 2003; Weiß, 1999).

Davis (1980) referiu que o objetivo da Inteligência Artificial Distribuída está relacionado com a resolução de problemas de situações que não são possíveis de solucionar com apenas uma entidade munida de inteligência artificial. Por seu lado, Nilsson (1981) apontou que esta estava relacionada com o tipo de resolução de problemas para o qual a computação ou a inferência estão distribuídas fisicamente ou logicamente. Mais tarde, Weiß (1999) referiu que o objetivo a longo prazo da Inteligência Artificial Distribuída consistem em desenvolver mecanismos e métodos que permitam aos agentes interagir tão bem como os humanos (ou até melhor) e perceber as interações entre entidades inteligentes, quer estas sejam computacionais, humanas, ou ambas. Reis (2003) apontou ainda que a Inteligência Artificial Distribuída se concentra na resolução de problemas onde diversos agentes resolvem subtarefas e comunicam numa linguagem de alto-nível.

Tradicionalmente, como é ilustrado na Figura 4.1, a Inteligência Artificial Distribuída está dividida em duas áreas (Panait e Luke, 2005; Reis, 2003; Weiß, 1999). Além dos Sistemas Multiagente, a Resolução Distribuída de Problemas relaciona-se com a decomposição e distribuição de um processo de resolução de problemas entre múltiplos nós e com a construção coletiva de uma solução para o problema. Os Sistemas Multiagente destacam os comportamentos comuns dos agentes com algum grau de autonomia e a

complexidade que surge das suas interações. Estes sistemas preocupam-se com a coordenação dos comportamentos de uma determinada comunidade de agentes, de modo a partilhar conhecimento, capacidades, e objetivos tendo em vista a resolução de problemas complexos.

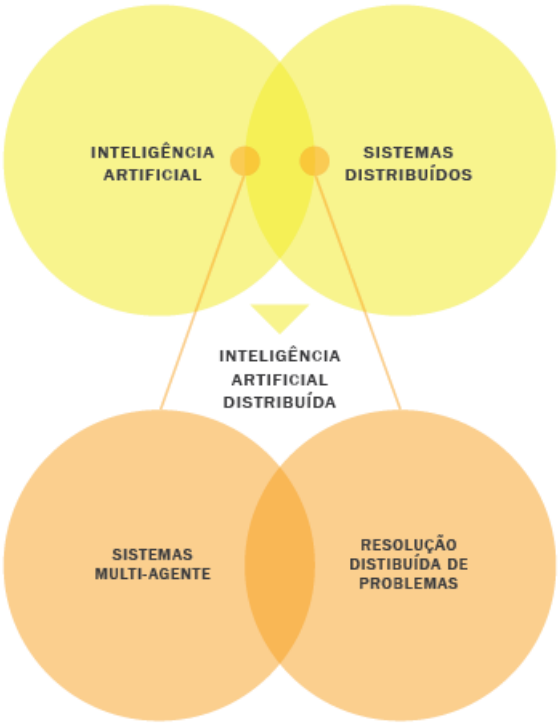

Figura 4.1 – Inteligência Artificial Distribuída (adaptada de (Reis, 2003)

Para se usar corretamente um Sistema Multiagente é necessário compreender o que se entende por agente, quais as suas características, e qual o modo como os agentes se organizam na presença de outros agentes, dando origem aos modelos multiagente. Assim, nesta secção serão descritos os conceitos de "agente" e de "Sistema Multiagente", com alguns exemplos de aplicações.

4.2. O agente

O termo "agente" atingiu o seu auge a partir dos anos 90, com a expansão da Internet nomeadamente por terem aparecido novas aplicações nas quais o conceito de agente aparece como a resposta mais adequada em termos tecnológicos (Reis, 2003). Mas é possível afirmar com algum grau de certeza que o campo dos agentes existe desde o encontro de 1956, em Dartmouth, onde John McCarthy introduziu o termo "Inteligência Artificial" (Wooldridge, 2002).

Um agente é um sistema computacional capaz de tomar ações independentes em nome do seu utilizador ou dono (Wooldridge, 2002). Pode ser visto como uma entidade que percebe e age sobre o seu ambiente autonomamente, pelo menos parcialmente, dependendo da sua própria experiência. Como uma entidade inteligente, um agente opera de forma flexível e racional numa variedade de circunstâncias que ocorrem no seu ambiente. A flexibilidade e racionalidade comportamental são alcançadas através de processos chave tais como a resolução de problemas, o planeamento, a tomada de decisão, e a aprendizagem. Sendo uma entidade de interação, um agente pode ser afetado nas suas atividades por outros agentes e até por seres humanos (Weiß, 1999).

A investigação na área dos agentes autónomos inspirou-se nas áreas científicas da Inteligência Artificial, Engenharia de Software, Sistemas Distribuídos e Redes de Computadores, Sociologia, e Teoria dos Jogos e Economia (Tabela 4.1).

Tabela 4.1 – Inspiração da investigação em agentes autónomos (Reis, 2003)

Área científica	Aspetos de inspiração
Inteligência Artificial	Micro-aspetos, como a resolução de problemas, raciocínio lógico, representação e utilização de conhecimento, planeamento, aprendizagem, etc.
Engenharia de Software	O agente como uma abstração, programação orientada por agentes.
Sistemas Distribuídos e Redes de Computadores	Arquiteturas de agentes, Sistemas Multiagente, comunicação e coordenação.
Sociologia	Macro-aspetos como a formação de sociedades virtuais e a interação entre agentes.
Teoria dos Jogos e Economia	Negociação, resolução de conflitos, mecanismos de mercado.

No entanto, continua a haver alguma controvérsia que envolve esta área de investigação (Reis, 2003). Este facto deve-se essencialmente a:

- Inexistência de um paradigma de programação bem definido para sistemas distribuídos;
- O termo “agente” ser comummente utilizado para descrever *software* em geral devido às definições vagas e contraditórias de que é alvo;
- O paradigma dos agentes tentar resolver o problema da assunção do “mundo fechado” na orientação a objetos;
- Ao interesse da comunicação social no assunto que resultou na extrapolação da área científica para o público em geral, sem que o seu significado fosse corretamente explicado.

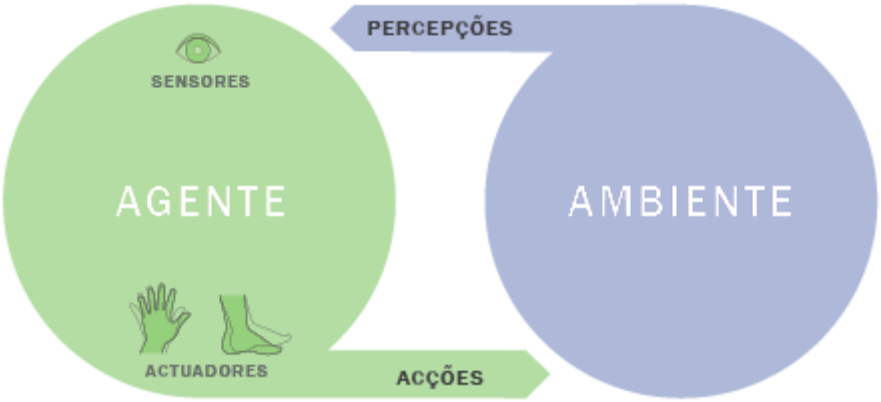

Figura 4.2 – Esquema típico de um agente (adaptada de (Reis, 2003))

Um agente deve possuir sensores e atuadores apropriados ao seu ambiente e à execução das tarefas para as quais foi projetado. A Figura 4.2 apresenta um esquema típico de um agente. Um ser humano apercebe-se do seu ambiente através dos 5 sentidos (visão, audição, olfato, paladar e tato) e age nesse ambiente utilizando os seus atuadores: braços, pernas, cordas vocais, etc. Os sensores de um agente robótico podem incluir câmaras (em analogia aos olhos), microfones (em analogia aos ouvidos), sensores de proximidade (incluindo, entre outros, infravermelhos e ultrassons), sensores de tato e aceleração, entre outros (Reis, 2003).

Nos agentes de *software* é mais difícil definir o que são os seus sensores e atuadores (Reis, 2003). Por exemplo, para um agente que jogue uma partida de xadrez, os sensores permitir-lhe-ão determinar o posicionamento das peças no tabuleiro e os atuadores terão a capacidade de agir no tabuleiro realizando jogadas.

De seguida, serão descritas algumas definições do termo “agente” bem como algumas das suas características.

4.2.1. Definições de agente

O termo “agente” apresenta, entre outras, as seguintes definições[1]:

- Que atua;
- Pessoa encarregada de praticar certas operações materiais ou representar os interesses de outrem;
- Aquilo que age, produz um efeito; o que origina (algo);

[1] http://www.infopedia.pt

Nas Ciências da Computação, a definição do termo "agente" não é consensual (Reis, 2003; Weiß, 1999; Wooldridge, 2002), sendo no entanto possível encontrar na literatura algumas definições que o caracterizam com proximidade (Tabela 4.2).

Com base nestas definições, é já possível perceber algumas características que devem estar associadas a um agente, como, por exemplo, a capacidade sensorial sobre o ambiente envolvente, a capacidade de agir e reagir sobre esse ambiente, a autonomia, as capacidades sociais para interação entre os agentes, entre outras, todas elas descritas mais pormenorizadamente na subsecção 4.2.2.

Tabela 4.2 – Definições do termo "agente"

Autor	Definição
Minsky (1986)	"Chamarei sociedade da mente a um esquema no qual cada mente é feita com muitos pequenos processos, chamados agentes. Cada agente só pode fazer coisas simples que não exijam qualquer mente ou pensamento, no entanto quando juntamos tais agentes em sociedades e de modos especiais tal conduzirá à verdadeira inteligência."
Brustolini (1991)	"Os agentes autónomos são sistemas capazes de realizarem autonomamente ações com sentido num mundo real."
Coelho (1995)	"Os agentes, para sobreviver, são forçados a possuir capacidades de tomada de decisão, estratégica e previsional, de coordenar as suas ações entre si e de enfrentar tarefas complicadas de forma efetiva."
Smith *et al.* (1994)	"Um agente é uma entidade de *software* persistente, dedicada a uma tarefa específica. O conceito de persistência distingue o agente de uma sub-rotina. Os agentes têm as suas próprias ideias de como cumprir as tarefas e têm as suas próprias agendas. Os agentes são entidades de finalidade específicas e não multifuncionais."
Maes (1995)	"Os agentes habitam ambientes dinâmicos e complexos, sentem e agem autonomamente nesses ambientes e realizam um conjunto de objetivos ou tarefas para as quais foram concebidos."
Wooldridge e Jennings (1995)	"Os agentes são sistemas baseados em *hardware* ou *software* com as seguintes propriedades: autonomia, capacidade social, reação, e pro-atividade."
Franklin e Graesser (1997)	"Um agente autónomo é um sistema que está contido, faz parte de um ambiente e atua sobre este ao longo do tempo, possuindo a sua própria agenda."
Jennings *et al.* (1998)	"Um agente é um sistema computacional, integrado num qualquer ambiente, capaz de ações flexíveis e autónomas de modo a realizar os objetivos"
Bernon *et al.* (2005)	"Um agente é uma entidade autónoma com uma dimensão reativa, proactiva, e social."
Panait e Luke (2005)	"Um agente é um mecanismo computacional que exibe um alto grau de autonomia, executa ações no seu ambiente baseado em informação (sensores, atuadores) recebida a partir do ambiente."
Russell e Norvig (2010)	"Um agente é algo que sente o ambiente onde existe através de sensores e atua através de atuadores."

4.2.2. Características dos agentes

Após se ter analisado várias definições de agente encontradas na literatura, é possível então, resumidamente, definir um agente como *"um sistema que age de forma autónoma dentro de um determinado meio e que procura satisfazer os seus próprios objetivos, através de sensores e atuadores"* (Pereira, 2009). Assim, um agente pode apresentar algumas ou mesmo todas as propriedades seguintes.

A **autonomia** de um agente permite que este decida e controle as suas próprias ações para satisfazer os seus objetivos (Bellifemine *et al.*, 2007; Bernon *et al.*, 2005; Brustolini, 1991; Franklin e Graesser, 1997; Jennings *et al.*, 1998; Maes, 1995; Panait e Luke, 2005; Reis, 2003; Weiß, 1999; Wooldridge, 2002; Wooldridge e Jennings, 1995). Alguns autores defendem que a autonomia aumenta com o aumento da pró-atividade, sem a necessidade de agirem em virtude de mudanças no ambiente ou a pedido de um humano ou outro agente (Wooldridge e Jennings, 1995). Reis (2003) referiu que a autonomia nunca pode ser totalmente obtida. O agente tem de ser criado e colocado em funcionamento originalmente por um humano (ou por um outro agente). O pressuposto que o funcionamento de um agente não terá um término também não é totalmente válido pois o agente terá um tempo de vida limitado e um final de operação. Por outro lado, embora seja essencial para poder ser considerado um agente, que o seu funcionamento seja efetuado sem operação direta de um humano, usualmente a interação com o humano é desejável. Assim, é habitual a construção de agentes que embora se possam comportar de forma autónoma, são também capazes de acatar ordens ou instruções de humanos, se tal for desejável.

A **reatividade** permite que o agente possa sentir e reagir mediante as alterações que vão ocorrendo no seu meio ambiente (Bellifemine *et al.*, 2007; Bernon *et al.*, 2005; Reis, 2003; Weiß, 1999; Wooldridge, 2002; Wooldridge e Jennings, 1995). Embora a reatividade seja uma característica desejável, um agente puramente reativo reage consecutivamente às mudanças no ambiente sem procurar atingir os seus objetivos de médio ou longo prazo (Reis, 2003). No entanto, construir agentes puramente reativos não é muito difícil (Reis, 2003; Weiß, 1999). Scheutz e Schermerhorn (2003) efetuaram uma comparação entre tipos de agente, tendo em conta a sua reatividade, chegando à conclusão de que os agentes deliberativos com capacidades de previsão obtiveram os melhores resultados.

A **pró-atividade** possibilita que o agente se oriente através de determinados objetivos e não atue apenas para responder ao ambiente, de modo a tomar a iniciativa para realizar tarefas mesmo sem serem explicitamente dadas pelo utilizador (Bellifemine *et al.*,

2007; Bernon *et al.*, 2005; Reis, 2003; Weiß, 1999; Wooldridge, 2002; Wooldridge e Jennings, 1995). Tal como Reis (2003) referiu, a pró-atividade é simples de conseguir em sistemas funcionais. No entanto, esta simplicidade só se verifica se for considerado que o ambiente é estático, i.e., não muda durante a execução de um dado procedimento ou função. Para além disso, o agente tem que dispor de toda a informação que necessita para executar esse procedimento ou função, sem incerteza no ambiente. No entanto, estas assunções não são válidas para a maioria dos ambientes, pois para ambientes dinâmicos e não totalmente acessíveis, os agentes têm de ser capazes de reagir às mudanças no ambiente e questionar se os objetivos originais ainda são válidos, em face das mudanças no ambiente enquanto executam um dado procedimento. Isto significa que os agentes têm de ser reativos e consequentemente capazes de reagir rapidamente às mudanças no ambiente.

A **capacidade social** permite que os agentes comuniquem com outros agentes e talvez com seres humanos, cooperando, concorrendo ou competindo para a obtenção de soluções (Bellifemine *et al.*, 2007; Bernon *et al.*, 2005; Franklin e Graesser, 1997; Reis, 2003; Weiß, 1999; Wooldridge, 2002; Wooldridge e Jennings, 1995). Na cooperação os agentes comunicam entre si e trabalham em conjunto com a finalidade de concluírem tarefas de interesse comum. Na competição vários agentes tentam obter o que apenas alguns deles podem ter (Weiß, 1999).

A **capacidade sensorial** é a capacidade de possuir sensores para captar informação relativa ao ambiente envolvente, que normalmente é dinâmico e imprevisível (Dunin-Keplicz e Verbrugge, 2010; Panait e Luke, 2005; Russell e Norvig, 2010; Weiß, 1999; Wooldridge, 2002). Estes sensores podem ser físicos ou virtuais, dependendo do tipo de agente (Reis, 2003).

Um agente com **persistência** é um agente que existe ao longo do tempo (Franklin e Graesser, 1997; Reis, 2003; Smith *et al.*, 1994). O conceito de persistência distingue o agente de uma sub-rotina. Os agentes têm as suas próprias ideias de como cumprir as tarefas e têm as suas próprias agendas (Smith *et al.*, 1994).

A **aprendizagem** possibilita que os agentes mudem as suas ações mediante experiências anteriores, de modo a se adaptarem ao ambiente (Bellifemine *et al.*, 2007; Reis, 2003; Wooldridge, 2002). Tal como os seres humanos são capazes de aprender ao longo do tempo à medida que vão interagindo com o meio ambiente, acedendo a mais informação e conhecimento, também os agentes devem ser capazes de evoluir, ou seja, aprender com base nas suas experiências anteriores (Pereira, 2009).

A **mobilidade** permite que um agente seja capaz de se movimentar de um local para outro, de modo a satisfazer os seus objetivos (Bellifemine *et al.*, 2007; Reis, 2003). Um

agente de *software* com esta característica pode, num determinado momento, abandonar o ambiente computacional onde operava e passar para outro ambiente computacional distinto, através de uma rede de computadores (Reis, 2003). A mobilidade pode também ser considerada numa outra perspetiva, mais abrangente, no caso dos agentes físicos (agentes que possuem um corpo). Neste caso, a mobilidade refere-se à capacidade do agente físico se deslocar fisicamente no seu ambiente, movimentando-se de uma localização para outra (Reis, 2003).

A **flexibilidade** é a característica que permite a um agente alterar facilmente as tarefas em execução. Assim, a ordem das tarefas a executar não precisa de ser pré-determinada (Jennings *et al.*, 1998).

O **carácter** ou personalidade de um agente permite que este tenha comportamento emocional de modo a que seja credível (Bellifemine *et al.*, 2007; Reis, 2003).

A **inteligência** de um agente implica várias das outras características descritas pois permite que um agente tenha capacidade de raciocínio autónomo, de planear as suas ações, de corrigir os erros e reagir a situações não esperadas, sendo capaz de se adaptar, aprender e otimizar o seu desempenho (Reis, 2003; Weiß, 1999).

A classificação de agente é feita com base nas características apresentadas. Não é usual encontrar um agente que possua todas as características mencionadas, embora, tal como já referido, algumas sejam fundamentais para o estabelecimento do conceito de agente. Além da autonomia, as que se têm revelado mais importantes são a reatividade, a pró-atividade e a capacidade social.

4.3. Sistemas baseados em agentes

Um Sistema Multiagente é constituído por vários agentes, interagindo entre si tipicamente através da troca de mensagens. Para que esta interação seja bem-sucedida, os agentes necessitam de ter a capacidade de cooperar e negociar entre si, da mesma forma que os seres humanos cooperam e negoceiam no seu dia-a-dia (Reis, 2003; Wooldridge, 2002).

A investigação em Sistemas Multiagente progrediu quase independentemente da investigação em agentes únicos até ao início da década de 1990. Apesar da noção de "agente" como um sistema isolado ser visível na literatura clássica de Inteligência Artificial, a noção de "Sistema Multiagente" só começou a ganhar proeminência a partir dos anos 1980.

Até meados dos anos 1980, a ênfase era dada à Resolução Distribuída de Problemas (Wooldridge, 2002).

O interesse nos Sistemas Multiagente cresceu muito rapidamente na primeira metade da década de 1990, devido à divulgação da Internet, que passou de uma mera ferramenta académica a algo usado diariamente em todo o globo para comércio e lazer. Com o crescimento e proliferação da Internet, no final dos anos 1990, chegou o comércio eletrónico que se percebeu ter um potencial lucrativo muito grande para a aplicação de Sistemas Multiagente. Desde então, os investigadores em Sistemas Multiagente começaram cada vez mais a procurar domínios realistas para os quais pudessem desenvolver este tipo de sistemas (Wooldridge, 2002). Por outro lado, os Sistemas Multiagente constituem uma metáfora natural para perceber, construir ou simular um vasto conjunto de sociedades artificiais (Reis, 2003; Wooldridge, 2002). Luck *et al.* (2005) referem mesmo que estes sistemas podem ser o próximo grande passo na evolução da computação, tal como a Programação Orientada a Objetos o foi.

Os sistemas baseados em agentes podem ser classificados em duas categorias (Stone e Veloso, 1998): os **Sistemas** centralizados **de Agente Único** e os **Sistemas Multiagente**. Nos primeiros, existe um agente com a função de tomar decisões. Os outros constituintes do sistema trabalham com base nessas decisões. 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 seus próprios objetivos, ações e conhecimento. Num ambiente com um único agente, nenhuma outra entidade é reconhecida como agente, tudo são elementos que formam o ambiente. No segundo tipo, os Sistemas Multiagente diferem dos Sistemas de Agente Único pelo facto de existirem vários agentes que modelam os objetivos e ações dos outros agentes. Num cenário multiagente típico, existem interações entre os agentes, isto é, comunicação entre si. Do ponto de vista individual de um agente, a principal diferença reside na possibilidade da dinâmica do ambiente ser determinada por outros agentes. Deste modo, todos os Sistemas Multiagente são considerados dinâmicos, isto é, o ambiente pode ser alterado enquanto um determinado agente atua.

Panait e Luke (2005) definiram um ambiente multiagente como aquele no qual *"existe mais do que um agente, que interagem uns com os outros, onde existem restrições nesse ambiente tal que os agentes não possam, num dado momento, saber tudo o que os outros agentes sabem sobre o mundo (incluindo os estados internos dos outros agentes)"*. Estas restrições são, na opinião de Panait e Luke (2005), importantes para a noção da definição

de problema de um Sistema Multiagente. De outra forma, os agentes distribuídos podem agir sincronizados, sabendo exatamente em que situações estão os outros agentes e quais os comportamentos que eles irão prosseguir. Esta "omnisciência" permite aos agentes agir como se fossem realmente meros apêndices de um controlador-mestre único. Adicionalmente, se o domínio não exigir interações, então este será decomposto em tarefas separadas e totalmente independentes, cada uma resolvida por cada agente único.

Como exemplo ilustrativo, considere-se a aplicação de exploração cooperativa, onde múltiplos agentes robôs têm a tarefa de descobrir amostras de rochas e levá-las para os locais respetivos. Na sua forma mais simples, este é um domínio de problemas que pode ser resolvido por um único robô, e múltiplos robôs (totalmente independentes) meramente a ajudar no esforço. O problema torna-se mais complexo quando os robôs podem, através de interações entre eles, sugerir gradualmente aos outros agentes boas regiões para explorar. Adicionalmente, se todos os robôs souberem instantaneamente todas as descobertas de rochas, e além disso souberem as áreas que os outros agentes escolheram para explorar, eles podem ser codificados para operar de forma similar a uma configuração mestre-súbdito (Panait e Luke, 2005).

Os Sistemas Multiagente podem então ser utilizados em problemas cuja complexidade enquanto um todo os torna intratáveis para os seres humanos ou para um único sistema de agente único. O recurso a técnicas de decomposição de problemas em subproblemas permite distribuir a computação, e os problemas resultantes, mais simples, são mapeados num conjunto de agentes distintos que interagem de forma a obterem soluções globais satisfatórias.

4.3.1. Motivação

A principal motivação da utilização dos Sistemas Multiagente está relacionada com o facto de grande parte dos problemas a resolver serem inerentemente distribuídos (Reis, 2003). Além disso, Reis (2003) referiu que:

- a dimensão do problema pode ser demasiado elevada para poder ser resolvido por um único agente;
- pode ser necessário permitir a interconexão e interoperação de múltiplos sistemas legados ("*legacy*");
- pode ser desejável providenciar uma solução natural para problemas geograficamente e/ou funcionalmente distribuídos;

- pode ser importante fornecer soluções para problemas em que os peritos, os conhecimentos ou as informações necessárias para a sua resolução se encontram distribuídos;
- uma interface cooperativa homem-máquina mais natural em que ambos funcionam como agentes no sistema pode ser importante;
- pode ser necessário oferecer uma maior clareza e simplicidade conceptual de projeto.

Os Sistemas Multiagente oferecem potencialmente uma maior rentabilidade de recursos para problemas onde o conhecimento é distribuído (Reis, 2003). Existem desta forma, várias razões adicionais para a utilização de um Sistema Multiagente (Stone e Veloso, 2000; Weiß, 1999):

- O domínio do problema assim o exige, por exemplo devido à distribuição espacial dos intervenientes;
- O paralelismo, atribuindo diferentes tarefas a diferentes agentes, de forma a que a execução seja mais rápida;
- A robustez, pois são utilizados diferentes agentes não existindo desta forma um ponto único de falha no sistema;
- A escalabilidade, permitindo o aumento dos agentes intervenientes num determinado sistema aberto;
- A simplificação de tarefas individuais de programação, dividindo o problema global em vários subproblemas;
- O estudo da inteligência individual e do comportamento social, pois os Sistemas Multiagente permitem a interoperacionalidade entre os agentes;
- A manutenção da privacidade da informação e conhecimentos de cada agente.

Em algumas circunstâncias a própria natureza do problema pode necessitar de um Sistema Multiagente, tal como no caso da marcação distribuída de reuniões, onde os agentes autónomos se encontram, realmente, geograficamente distribuídos (Reis, 2003).

A utilização de Sistemas Multiagente na resolução de problemas de Inteligência Artificial apresenta diversos benefícios, nomeadamente uma maior rentabilidade de recursos para problemas onde o conhecimento ou atividade é distribuído (Reis, 2003):

- Resolução mais rápida de problemas devido ao processamento concorrente;
- Diminuição da comunicação devido ao processamento estar localizado junto à fonte de informação e a comunicação ser realizada a alto-nível;

- Aumento da flexibilidade e escalabilidade resultantes da possibilidade de interconexão de múltiplos sistemas com arquiteturas distintas;
- Aumento da fiabilidade devido à inexistência de um ponto singular de falha;
- Aumento da capacidade de resposta devido aos sensores, sistemas de processamento e atuadores estarem localizados em conjunto, no interior dos agentes;
- Facilidade acrescida de desenvolvimento de sistemas devido à modularidade resultante da decomposição dos problemas e da decomposição dos sistemas em agentes semiautónomos.

4.3.2. Modelos de Sistemas Multiagente

Os Sistemas Multiagente podem ser organizados de várias formas, desde hierarquias, passando por equipas, até federações. (Horling e Lesser, 2004) descreveram vários modelos de Sistemas Multiagente, descritos de seguida.

Os **modelos hierárquicos** são os mais utilizados na especificação de Sistemas Multiagente. Nestes modelos, os agentes organizam-se numa estrutura em árvore, onde os agentes de nível superior têm controlo sobre os agentes de nível inferior e onde as interações normalmente só ocorrem entre agentes interligados por arcos ou ramos da árvore. Estes modelos são indicados à modelação dos mais variados tipos de problemas. No entanto, podem ser considerados frágeis já que uma falha num dos níveis superiores pode acarretar uma falha global do sistema. Outro problema é a suscetibilidade a constrangimentos (*bottlenecks*) se a gestão da informação não for bem concretizada.

Os **modelos holónicos** representam sistemas em que cada sistema é constituído por subsistemas que por sua vez são constituídos por infra sistemas. Cada elemento de um sistema holónico é um *holon*, que resulta da combinação da palavra grega *holos*, que significa “todo”, com o sufixo *on* da língua inglesa, que sugere “parte”, como em *proton* ou *neutron*. Assim sendo, *holon* passa a designar o todo e a parte, implicando que tenha uma natureza recorrente. Estes modelos têm uma estrutura de controlo semelhante à dos modelos hierárquicos, tendo a grande diferença na autonomia dos *holons* em relação ao modo como os seus “subordinados” realizam tarefas. A desvantagem deste modelo, para além do desenho lógico dos níveis em holarquias (“*holarchy*”, em inglês), prende-se com a ausência de informação dos níveis superiores sobre o modo como a tarefa é realizada, o que implica uma dificuldade acrescida na previsão do desempenho do sistema.

Os **modelos de aliança** têm origem na teoria de jogos, sendo que as alianças entre agentes só existem enquanto não é satisfeito um objetivo comum, e deixam de existir quando esse objetivo é atingido. Uma aliança de agentes pode ser tratada como uma entidade única e autónoma e cada agente dessa aliança tem a responsabilidade de cooperar e coordenar as atividades de modo a que o objetivo seja alcançado. A vantagem destes modelos é a suposição de que o todo é maior do que a soma das partes, o que pode significar que uma aliança pode receber uma tarefa que um agente único não teria capacidade para realizar, e dessa forma aumentar a utilidade de todos os agentes da aliança. A desvantagem prende-se com o modo de formação das alianças, sobretudo em ambientes dinâmicos, uma vez que a agregação de agentes pode ser difícil por estes poderem estar associados a outras alianças.

Os **modelos em equipa** consistem num número de agentes cooperantes que trabalham conjuntamente de forma a alcançar um objetivo comum. Em relação ao modelo anterior, a diferença consiste em alcançar o máximo de utilidade do sistema (equipa) e não a utilidade individual, e é esperado que todos os agentes atuem em prol do objetivo da equipa. Como nas alianças, uma das vantagens de trabalhar em equipa é a possibilidade de resolver problemas de grande dimensão. Uma característica única destes modelos é a existência de redundância e a capacidade de resolver restrições globais. As grandes desvantagens são a interação entre todos os agentes e a constante dispersão dos diferentes estados em que os agentes se encontram, uma vez que o número de comunicações aumenta significativamente em relação a outros modelos.

Os **modelos de congregações** são semelhantes a alianças e equipas, mas, ao contrário destes modelos, as congregações são formadas por agentes com características semelhantes ou complementares de modo a facilitar o processo de colaboração. Os agentes não têm um objetivo fixo mas sim um conjunto de capacidades ou requisitos que os leva a congregar. Estes agentes tentam maximizar a sua utilidade individual sendo esta necessidade que conduz à procura de membros que potencialmente podem contribuir para atingir os seus objetivos (estimulo à congregação). No entanto é necessário que ambos os agentes retirem benefícios substanciais da congregação pois há um elevado tempo e energia despendida na procura e formação da congregação.

Os **modelos em sociedade** são inerentes a sistemas abertos onde várias classes de agentes podem interagir, ingressar e abandonar a sociedade de livre vontade enquanto essa persiste. Uma vez que os agentes têm objetivos e capacidades diferentes, a sociedade tem de ser capaz de estabelecer regras, normas e convenções de forma a fornecer um nível de

consistência e comportamento necessário à coabitação. O grande problema destes modelos é a sua complexidade no estabelecimento de regras da sociedade, pois estas têm de promover um equilíbrio entre flexibilidade que promove a obtenção dos objetivos e as restrições necessárias à vida em sociedade.

Os **modelos em federação** admitem um grupo de agentes que concedem alguma autonomia de modo a poderem delegar num agente a representação da organização, chamado de facilitador, mediador ou intermediário. Os membros da federação interagem apenas com este mediador que é responsável pela comunicação de pedidos, estados e descrições do grupo com o exterior. A desvantagem com estes modelos é a tendência de afunilar demasiadas funções nos mediadores, originando constrangimentos (*bottlenecks*) no sistema.

Nos **modelos de mercado** existem três tipos de agentes: os compradores, os vendedores e os mediadores. Os compradores fazem propostas para utilização de recursos, realização de tarefas e aquisição de serviços ou bens enquanto os vendedores colocam um preço nas suas mercadorias e o mediador (que pode ser o vendedor) é responsável pelo leilão e determinação do vencedor. Estes modelos têm um controlo semelhante aos modelos federativos, onde um agente é responsável pela coordenação de um grupo de agentes, mas no entanto os participantes são competitivos entre si e não cedem autonomia aos mediadores, embora confiem nas suas deliberações. Ao tirar partido das regras de mercado, aumenta-se a justiça das interações, desde que os agentes se comportem condignamente, ou seja, que não burlem o mercado. Uma forma de desencorajar este tipo de comportamento é a utilização de anonimato e comunicações seguras aliadas a estratégias dissuasoras.

Os **modelos** baseados em **organizações matriciais** recorrem a uma estrutura em que um agente ou uma equipa de agentes são geridos por mais do que um agente-gestor. Baseiam-se no modo como os humanos agem, pois estes sofrem influências de diversas entidades ao mesmo tempo (família, emprego, etc.), e é uma forma de os agentes partilharem capacidades entre si, esperando-se que as várias influências acabem por gerar um benefício alargado. O agente tem de ter autonomia e raciocínio de modo a resolver conflitos entre os vários objetivos dos gestores. Uma maneira de resolver conflitos pode ser a atribuição de funções de utilidade ao binómio entre o trabalho a realizar e o benefício desse trabalho para outras entidades.

Por último, os **modelos compostos** referem-se a combinações dos modelos acima descritos, e possuem as vantagens e desvantagens dos tipos de modelos adotados.

4.3.3. Coordenação em Sistemas Multiagente

Uma das principais características dos Sistemas Multiagente é a comunicação, uma vez que os agentes necessitam de comunicar uns com os outros, com os utilizadores, ou até com os recursos do sistema, de modo a cooperarem, colaborarem e negociarem com o objetivo de resolver problemas (Bellifemine *et al.*, 2007; Dunin-Keplicz e Verbrugge, 2010).

Os agentes interagem uns com os outros através do uso de linguagens de comunicação especiais, designadas de linguagens de comunicação de agentes e que fornecem uma separação entre atos comunicativos e linguagem de conteúdo (Bellifemine *et al.*, 2007). No entanto, para que os agentes possam trabalhar em conjunto de forma coerente é necessário mais do que a capacidade de comunicarem entre si. Sendo os Sistemas Multiagente inerentemente distribuídos, torna-se necessário coordenar os agentes de modo a conseguir gerir as suas interações e dependências de atividades (Reis, 2003; Weiß, 1999).

A coordenação pode ser definida como *"o ato de trabalhar em grupo de forma harmoniosa"* (Malone e Crowston, 1994). No entanto, tal como acontece com a definição do conceito de agente, também esta definição não é partilhada por toda a comunidade científica (Reis, 2003). Desta forma, outros autores propuseram diversas definições para o termo "coordenação" (Tabela 4.3).

Tabela 4.3 – Definições do termo "coordenação"

Autor	Definição
Singh (1992)	"A integração e ajustamento harmonioso dos esforços individuais no sentido de alcançar um objetivo mais amplo"
Jennings (1996)	"Processo pelo qual um agente raciocina acerca das suas ações locais e das ações (antecipadas) dos outros para tentar assegurar que a comunidade atue de maneira coerente"
Nwana *et al.* (1996)	"Processo no qual diferentes agentes se envolvem para garantir que a sua comunidade age de forma coerente"
Weiß (1999)	"Propriedade de um sistema de agentes que realizam atividades em ambientes compartilhados"
Reis (2002)	"O ato de trabalhar em conjunto de forma harmoniosa no sentido de atingir um acordo ou objetivo comum"
Luck *et al.* (2005)	"Processo para garantir que as ações de atores (agentes) independentes num ambiente são coerentes de alguma forma"

Bellifemine *et al.* (2007) referiram que os agentes podem ter de coordenar porque os seus objetivos podem causar conflitos entre as suas ações, os seus objetivos podem ser interdependentes, os agentes podem ter diferentes capacidades e conhecimento, e ainda porque os objetivos dos agentes podem ser atingidos mais rapidamente se os diferentes agentes trabalharem em conjunto. Por seu lado, Jennings (1996) afirmou que os agentes podem coordenar porque existem dependências nas ações nos agentes, ou porque existe a necessidade que o conjunto de agentes respeite restrições globais, ou ainda porque nenhum agente individualmente tem recursos, informação ou capacidade suficiente para executar a tarefa ou resolver o problema completo.

Mesmo quando é possível que os agentes individuais trabalhem independentemente dos restantes, a coordenação pode aumentar a eficiência do sistema (Reis, 2003). Desta forma, (Nwana *et al.*, 1996), adicionaram duas razões para a necessidade de coordenar agentes num sistema: para aumentar a eficiência e/ou para prevenir a anarquia e o caos.

É assumido que a coordenação em Sistemas Multiagente é feita em tempo de execução, i.e., que os agentes têm a capacidade de reconhecer os relacionamentos e, quando necessário, geri-los como parte das suas atividades. Isto contrasta com a situação mais convencional onde o programador tenta explicitamente prever quaisquer interações e programar o sistema para evitar interações negativas e para explorar potenciais interações positivas (Wooldridge, 2002).

Diversas metodologias de coordenação têm sido propostas por diferentes autores, dividindo-se principalmente em dois grupos (Bellifemine *et al.*, 2007; Madureira *et al.*, 2010a; Reis, 2003; Weiß, 1999):

- Metodologias aplicáveis em domínios cooperativos, contendo agentes que se preocupam pela utilidade global do sistema e não pela sua utilidade pessoal. Interessa assim estudar metodologias que permitam construir equipas de agentes.
- Metodologias aplicáveis em domínios competitivos, contendo agentes preocupados com o seu bem próprio, i.e., egoístas ("*self-interested*" em inglês). Os agentes não estão normalmente interessados no bem da comunidade mas sim, em obter a sua satisfação pessoal. A coordenação por negociação é a metodologia mais estudada.

4.3.3.1. Cooperação

A cooperação em contextos económicos e sociais pode ser vista como um relacionamento baseado na colaboração entre indivíduos ou organizações por forma a atingirem os seus objetivos (Madureira *et al.*, 2010a).

Na cooperação em Sistemas Multiagente, diversos agentes trabalham em conjunto e aproveitam a vasta coleção de conhecimento e capacidades de modo a atingir um objetivo comum, tentando alcançar como uma equipa o que individualmente não seriam capazes (Weiß, 1999). O interesse em Sistemas Multiagente onde agentes cooperativos trabalham em conjunto como equipas tem crescido significativamente nos últimos anos (Bellifemine *et al.*, 2007; Dunin-Keplicz e Verbrugge, 2010; Reis, 2003).

Gilbert (2005) definiu equipa de agentes de uma forma curiosa mas bastante simples: *"o termo 'equipa' lembra-me a ideia de um grupo social dedicado à procura de um objetivo particular e persistente: uma equipa desportiva a tentar ganhar, uma célula terrorista tentando atos terroristas, o grupo de trabalho a tentar atingir uma meta em particular"*.

Dunin-Keplicz e Verbrugge (2010) referiram que os agentes de equipas devem: trabalhar conjuntamente para atingir um objetivo comum, monitorizar constantemente o progresso do esforço da equipa como um todo, ajudarem-se mutuamente quando necessário, coordenar ações individuais para que não interfiram uns com os outros, comunicar sucessos e falhas (parciais) se necessário, evitar competição entre membros respetivamente à obtenção do objetivo comum.

Existem diferentes abordagens de como implementar a cooperação entre agentes, que podem ser divididas em dois grupos principais (Madureira *et al.*, 2010a):

- Cooperação entre agentes, onde cada agente é capaz de comunicar com os outros e expressar as suas necessidades para o grupo. Esta abordagem requer um alto nível de inteligência dos agentes, uma vez que estes devem ser capazes de analisar a tarefa a comunicar com cada agente para obterem a solução.
- Sistemas que utilizam um agente coordenador para analisar o problema e, baseados nas respetivas características, atribuir as tarefas a cada agente individualmente.

Um dos principais problemas da coordenação em Sistemas Multiagente é que a sua aplicação só é possível em domínios relativamente simples. De facto, verificam-se poucas metodologias aplicadas com sucesso em domínios dinâmicos, contínuos, inacessíveis, e

não determinísticos. Estes tipos de domínios abertos tornam-se ainda mais complexos quando existe uma necessidade de coordenação espacial dos agentes envolvidos, ou seja, quando estes possuem capacidades de mobilidade, tal como em cenários de guerra ou no futebol robótico. Desta forma, a criação de novas metodologias de coordenação de alto-nível de equipas de agentes é um tópico ainda pouco explorado dentro da investigação de coordenação em Sistemas Multiagente (Reis, 2003).

4.3.3.2. Competição/Negociação

A competição tem vindo a ser estudada em diversos campos incluindo a filosofia, a sociologia e a antropologia. Os psicólogos sociais, por exemplo, estudam a natureza da competição, investigando o desejo natural da competição e suas circunstâncias. Estudam também a dinâmica de grupo, para detetar como a competição emerge e quais os seus efeitos (Telser, 1987).

No caso da competição em Sistemas Multiagente, vários agentes trabalham uns contra os outros devido aos conflitos dos seus objetivos. Os agentes competitivos tentam maximizar os seus próprios benefícios à custa dos outros e assim o sucesso de uns implica o fracasso de outros (Weiß, 1999). Uma forma frequente de interação entre agentes competitivos é a negociação (Bellifemine *et al.*, 2007; Reis, 2003; Weiß, 1999). A negociação é um processo pelo qual uma decisão conjunta é atingida por dois ou mais agentes, com cada um a tentar atingir um objetivo individual (Weiß, 1999). A negociação pode ser definida como uma forma de interação na qual um grupo de agentes com interesses em conflito e desejo em cooperar tentam chegar a um acordo mútuo para a alocação de recursos escassos. Várias definições do termo "negociação" são propostas na literatura, nomeadamente: *"processo pelo qual um grupo de agentes comunica com outros para tentar alcançar acordos sobre assuntos de interesse comum"* (Viana, 2003).

Em geral, a literatura define os seguintes métodos de negociação:

- *Contract-Net Protocol* (Smith, 1980);
- Leilões (Wooldridge, 2002);
- Teoria de Jogos (Sandholm, 2000);
- Argumentação (Jennings et al., 2001).

O protocolo e a estratégia são os componentes principais de um mecanismo de negociação. O protocolo define as regras comuns entre os participantes no ato negocial. Em

geral, inclui um grupo de normas que restringem as propostas que os participantes podem fazer. A estratégia define as ações possíveis ou sequência de ações que um agente planeia fazer durante o processo de negociação (Jennings *et al.*, 2001).

A negociação pretende que a alocação de recursos seja aceite por todos os participantes. Uma vez que existem várias formas diferentes de acordos ou alocação de recursos, a negociação pode ser vista como uma pesquisa distribuída através de um espaço de acordos possíveis (Jennings, 2001). Normalmente a negociação é composta por um grupo de rondas, com todos os agentes a fazerem uma proposta em cada ronda. A proposta que cada agente faz é definida pela sua estratégia e deve estar concordante com o protocolo definido. A negociação termina quando for alcançado algum acordo (Wooldridge, 2002).

Um aspeto complexo no processo negocial é o número de agentes envolvidos e a forma como estes agentes interagem (Wooldridge, 2002). Existem três possibilidades: negociação um-para-um (em que um agente negoceia apenas com outro agente), negociação um-para-muitos (em que um agente negoceia com um número de agentes), e negociação muitos-para-muitos (em que vários agentes negoceiam simultaneamente com outros agentes).

Embora a negociação desempenhe um papel importante na coordenação de Sistemas Multiagente compostos por agentes competitivos, esta não é o mecanismo de coordenação mais aconselhável nas situações em que um conjunto de agentes partilha objetivos comuns. Nestes casos, a coordenação dos agentes pode ser realizada em grande parte por metodologias de cooperação uma vez que o conjunto de agentes partilha os mesmos objetivos e existem vantagens mútuas em cooperarem para os atingirem (Reis, 2003).

4.3.4. Áreas de aplicação

A tecnologia de agentes têm sido alvo de discussões extensas e investigação por parte da comunidade científica ao longo dos últimos anos, mas apenas recentemente se tem verificado a sua exploração em aplicações comerciais. Os Sistemas Multiagente têm sido usados numa grande diversidade de aplicações, desde pequenos sistemas para assistência pessoal até sistemas abertos, complexos e críticos para aplicações industriais (Bellifemine *et al.*, 2007; Jennings e Wooldridge, 1998; Reis, 2003; Weiß, 1999).

Uma das mais importantes áreas de aplicação dos Sistemas Multiagente é a **pesquisa e gestão da informação** (Bellifemine *et al.*, 2007; Decker e Sycara, 1997; Reis, 2003; Wooldridge, 2002). A Internet tem mostrado ser um domínio extremamente adequado para o uso de Sistemas Multiagente, devido à sua natureza intrinsecamente dinâmica e distribuída e ao grande volume de informação envolvida. Os agentes podem, por exemplo, procurar e filtrar estes volumes de informação (Klusch, 2001). Desta forma, as aplicações de agentes de pesquisa, recuperação e filtragem de informação na Internet constituem uma área em expansão no âmbito dos agentes autónomos. Neste contexto, têm também ganho relevância aplicações de agentes que possuem o objetivo de auxiliar um dado utilizador a realizar as suas tarefas mais repetitivas, normalmente designadas por assistentes pessoais (Reis, 2003). Aplicações nesta área podem ser encontradas em (Chen *et al.*, 2000; Chira e Dumitrescu, 2006; Jing-yan e Zhen, 2008; Shooter *et al.*, 2005; Wang *et al.*, 2011b)**.**

O grande crescimento do **Comércio Eletrónico** nas últimas décadas tem vindo a assumir um papel central em muitas organizações devido ao facto destas oferecerem oportunidades que melhoram significativamente a forma como as diferentes entidades envolvidas num processo de negócio interagem (Bellifemine *et al.*, 2007). Este facto, juntamente com a evidente aplicabilidade dos agentes a este domínio levaram ao aparecimento de inúmeras aplicações para Comércio Eletrónico envolvendo agentes (Bellifemine *et al.*, 2007; Reis, 2003; Weiß, 1999; Wooldridge, 2002). Exemplos de trabalhos de aplicação de agentes em Comércio Eletrónico podem ser encontrados em (Fang-Chang e Chien-Nan, 2002; Gleizes *et al.*, 2000; Jascanu, 2008; Qiang, 2009; Ruimei, 2011; Viamonte *et al.*, 2006).

O uso de Sistemas Multiagente na **robótica** é apelativo, uma vez que os robôs podem ser representados por agentes, com sensores e atuadores. Muitos sistemas robóticos usam Sistemas Multiagente e técnicas de planeamento distribuído para coordenar os diferentes robôs (Bellifemine *et al.*, 2007). Na maioria das aplicações, o controlo de robôs efetuado diretamente por humanos é, na prática, impossível e como tal o recurso a agentes autónomos é a solução mais aconselhável (Reis, 2003). Dois exemplos práticos são a exploração colaborativa, em que vários agentes dotados de autonomia e inteligência decidem as trajetórias a seguir e comunicam entre si para melhorar o desempenho da exploração, e o futebol robótico, em que uma equipa é composta por robots similares mas que cumprem funções diferentes (guarda-redes, defesas, avançados), podendo ser dotados de comportamentos reativos, e havendo planeamento estratégico e planeamento tático. Alguns trabalhos nesta área são (Bezek *et al.*, 2006; Burgard *et al.*, 2005; Estlin *et al.*, 2005;

Goldberg *et al.*, 2002; Kose *et al.*, 2005; Reis, 2003; Rostami *et al.*, 2005; Stone, 2000; Toriz *et al.*, 2009).

O **controlo de tráfego e transporte** é um domínio crítico caracterizado pela sua complexidade, mas com uma natureza distribuída (Bellifemine *et al.*, 2007; Weiß, 1999). Pode representar tráfego aéreo, em que existem agentes para representar voos, pistas de aterragem, fatores climatéricos, etc., ou mesmo tráfego urbano, em que existem agentes a representar semáforos, que podem negociar entre si para ajustar o fluxo de tráfego, ou até mesmo a cooperação de semáforos de modo a que os automobilistas não tenham de parar consecutivamente. Trabalhos recentes na área do controlo de tráfego e transporte podem ser encontrados em (Baiying e Wei, 2008; Balaji e Srinivasan, 2010; Desai *et al.*, 2011; Gorodetsky *et al.*, 2008; Kuyer *et al.*, 2008; Zhang *et al.*, 2009).

Os **sistemas de produção** caracterizam-se por um elevado grau de complexidade sendo distribuídos do ponto de vista físico, com as máquinas, linhas de fabrico, fábricas, etc., e lógico, com vários produtos, encomendas e ordens de fabrico. Os Sistemas Multiagente têm sido bastante usados nesta área, existindo várias implementações em **Escalonamento** (Shen e Norrie, 1999; Weiß, 1999). Alguns trabalhos com Sistemas Multiagente na área de Sistemas de Produção e Escalonamento podem ser encontrados em (Lin e Solberg, 1994; Madureira *et al.*, 2011a; Madureira *et al.*, 2009c; Maturana, 1997; Maturana *et al.*, 1999; Ouelhadj *et al.*, 2000; Pereira, 2009; Pereira e Madureira, 2013; Pereira *et al.*, 2013a; Qiao *et al.*, 2010; Shen *et al.*, 2000; Shen *et al.*, 2004).

Os **sistemas de telecomunicações** são outro domínio de aplicação onde os Sistemas Multiagente têm sido usados com sucesso (Bellifemine *et al.*, 2007; Weiß, 1999). De facto, este tipo de sistemas são representados por grandes redes distribuídas de componentes interligados que necessitam de ser monitorizados e geridos em tempo-real, e formam a base de um mercado competitivo onde as companhias de telecomunicações e fornecedores de serviços tentam distinguir-se dos seus competidores através do fornecimento de serviços melhores, mais rápidos e confiáveis (Bellifemine *et al.*, 2007). Exemplos de trabalhos nesta área são (Bonhomme *et al.*, 2010; Gâteau *et al.*, 2009; Greenwood *et al.*, 2006; Hayzelden e Bourne, 2001; Paolino *et al.*, 2011; Petric, 2008).

Outros exemplos de aplicações de Sistemas Multiagente encontram-se na área da **saúde** (Bellifemine *et al.*, 2007; Moreno e Nealon, 2004; Reis, 2003). Os Sistemas Multiagente têm vindo a ser propostos para resolver diversos problemas neste domínio, incluindo escalonamento e gestão de pacientes, gestão e acesso de informação médica, e suporte à decisão (Bellifemine *et al.*, 2007). Alguns trabalhos nesta área podem ser

encontrados em (De Meo *et al.*, 2011; Hudson e Cohen, 2002; Jiang e Tianfield, 2009; Lanzola e Boley, 2002; Shanhong *et al.*, 2010).

4.4. Sumário

Neste capítulo foram descritos os sistemas baseados em agentes, desde a descrição do conceito de agente até às suas características principais, tais como a autonomia, a reatividade, a aprendizagem, a inteligência, entre outras. Foram referidas as motivações no uso de Sistemas Multiagente bem como os vários modelos existentes. Por apresentar alguma relevância no âmbito deste trabalho, foi também apresentada uma descrição da coordenação em Sistemas Multiagente, com ênfase em cooperação e competição. Foram ainda apresentadas algumas áreas de aplicação destes sistemas.

Capítulo 5. Computação Autónoma

5.1. Introdução

No início do milénio, os avanços na tecnologia computacional e o aparecimento da comunicação em rede, principalmente a Internet, levaram as empresas a investir significativamente em infraestruturas e aplicações computacionais. Estes sistemas estavam sujeitos a falhas, dinamismo, sobrecargas, entre outros, devido ao crescimento exponencial da sua complexidade. As empresas investiam cada vez mais em manutenção do que em desenvolvimento (Parashar e Hariri, 2006), particularmente considerando as questões de ordem financeira e económica da altura.

Perante a previsão da chegada de um "ponto de rutura", em Outubro de 2001, Paul Horn (2001), vice-presidente da IBM Research, deu corpo e visibilidade a este problema, lançando um desafio com a designação de Computação Autónoma (*Autonomic Computing*). Em conjunto com a IBM, a comunidade científica tem vindo a contribuir para a nova geração de sistemas computacionais com, e.g., Computação Ubíqua, Computação baseada em Agentes, *Smart Grids*, entre outros (Lalanda *et al.*, 2013). Devido à complexidade dos desafios que têm surgido, a Computação Autónoma usufrui dos conhecimentos de diversas áreas, tais como engenharia de *software*, engenharia de sistemas, teoria de controlo, inteligência artificial, entre outros (Lalanda *et al.*, 2013).

Fitzgerald (2012) realçou que a segunda crise de software (*Software Crisis 2.0*) levou à necessidade da autogestão dos sistemas de software, devido à procura e exigência de dados digitais, juntamente com o grande volume de dados que são gerados através de dispositivos móveis, sensores e aplicações.

Hoje em dia, a implementação de sistemas com comportamentos de autogestão permanece um desafio, mas já é possível notar o sucesso da iniciativa de Computação Autónoma, através do foco em autonomia por parte de várias conferências e comunidades científicas (Lalanda *et al.*, 2013).

A Computação Autónoma é um paradigma da computação em que grandes sistemas informáticos são constituídos por mecanismos de manutenção quotidiana da própria aplicação embutidos, com o objetivo de a automatizar. Assim, as aplicações devem ser capazes de se acomodar, aproveitar e proteger das mudanças no ambiente e nos seus objetivos, de acordo com orientações de alto nível por parte dos seres humanos. Este novo

paradigma é inspirado no sistema nervoso central dos seres vivos, uma vez que muitas funções essenciais ao bem-estar e regularização do estado dos seres vivos não são desencadeadas conscientemente pela mente, por exemplo, o batimento do coração, o sistema digestivo, até a própria respiração, etc. No entanto, sem um sistema autónomo que faça a gestão destes mecanismos, ou o corpo deixaria de funcionar corretamente ou não era possível a concentração noutros aspetos da vida (Parashar e Hariri, 2006).

O objetivo da Computação Autónoma consiste em transferir a carga de tarefas de suporte, tais como configuração, manutenção e gestão de falhas, das pessoas para a tecnologia. Os Sistemas Autónomos são desenhados para serem responsáveis por operações intensivas, delegadas pelos profissionais, muitas das quais rotineiras e repetitivas (IBM, 2006). Computacionalmente, pretende-se um sistema pró-ativo, capaz de se autogerir, mediante determinadas regras e objetivos definidos pelo administrador, sem que seja totalmente excluída a intervenção humana no sistema.

A proposta da Computação Autónoma abrange capacidades desconhecidas nos produtos e ferramentas tradicionais. Não inclui apenas a capacidade de realizar ações autonomamente mas também de as fazer com base em habilidades inatas, de sentir e responder a mudanças. Um Sistema Autónomo não deve apenas executar regras mas continuamente normalizar e otimizar os sistemas em tempo real. Além disso, não deve apenas guardar e executar políticas mas também incorporar capacidades de autoaprendizagem e autogestão. A Computação Autónoma é um paradigma que propõe aliviar o peso de levar as tecnologias de informação para o futuro, através da passagem das tarefas mundanas para a tecnologia, de modo a libertar os seres humanos para o trabalho com mais impacto (IBM, 2006). Neste capítulo são abordados os diversos componentes de autogestão e os elementos básicos da arquitetura de um Sistema Autónomo.

5.2. Comportamentos de Autogestão

Um Sistema Autónomo é um sistema com capacidade de autogestão, podendo assim libertar os utilizadores de tarefas repetitivas, possibilitar a execução a tempo inteiro, e fornecer inteligência suficiente para que possam ser tomadas decisões para atingir um determinado objetivo. Este tipo de sistemas é capaz de se adaptar e sustentar perante mudanças na organização, tais como ordens de trabalho, alteração de componentes, alterações no ambiente, bem como reagir a falhas de *software* e *hardware*, sejam essas falhas acidentais ou provocadas (Kephart e Chess, 2003).

Um Sistema Autónomo deve (Ganek, 2006):

- Ter conhecimento, não só dos seus componentes, estado, capacidades, entre outros, mas também do contexto da sua atividade e noutros recursos da infraestrutura;
- Ser capaz de perceber e analisar as condições do ambiente. Inclui a capacidade para proactivamente perceber os componentes e serviços individuais, para procurar formas de melhorar as suas funções, assim como a aptidão de detetar mudanças no ambiente e perceber as suas implicações;
- Ser capaz de planear e efetuar mudanças através da alteração do seu estado, de modo a alterar outros componentes no ambiente.

Assim, foram definidas algumas propriedades, propostas inicialmente com o nome auto-* (em inglês, *self*-*) que caracterizam os sistemas que implementem o paradigma da Computação Autónoma (IBM, 2005; Kephart e Chess, 2003; Parashar e Hariri, 2006): Auto-Configuração (*Self-Configuring*), Auto-Otimização (*Self-Optimizing*), Auto-Recuperação (*Self-Healing*), e Auto-Proteção (*Self-Protection*).

5.2.1. Auto-Configuração

Os vários processos de instalação, configuração, e integração de sistemas complexos constituem um desafio, pelo que implicam perdas de tempo significativas e existe uma tendência natural para a ocorrência de erros, mesmo para pessoas experientes. Pretende-se então, com o uso dos Sistemas Autónomos, que o sistema seja capaz de configuração automática com base em políticas de alto-nível previamente definidas. Assim, quando um novo componente é inserido no sistema, este é incorporado de forma autónoma, tal como uma nova célula se incorpora no corpo humano, ou mesmo como uma pessoa se insere numa população.

O objetivo da Auto-Configuração passa por permitir que o sistema se adapte a determinados eventos imprevisíveis no ambiente através da alteração automática da sua configuração, tal como adicionar ou remover componentes ou recursos, ou proceder a alterações de *software* sem interrupções do serviço (Kephart e Chess, 2003).

5.2.2. Auto-Otimização

Com o número significativo de parâmetros de configuração que podem ser alterados e que tornam os sistemas difíceis de otimizar, a Auto-Otimização surge como a capacidade que permite aos Sistemas Autónomos procurarem continuamente formas de melhorar o seu desempenho, através da identificação e aproveitamento de oportunidades para se tornarem mais eficientes no seu desempenho e custo, e que proactivamente procurem melhorar as suas funções (Kephart e Chess, 2003). Permite também que um Sistema Autónomo faça a sua própria monitorização, planeamento e afinação de parâmetros, com base no conhecimento adquirido do contacto com o ambiente, para melhorar ativamente o seu desempenho e efetuar as escolhas mais apropriadas para alcançar um objetivo pretendido.

5.2.3. Auto-Recuperação

Os Sistemas Autónomos devem ser capazes de detetar e diagnosticar as raízes de problemas resultantes de falhas de *software* e *hardware* e proceder à reparação necessária para o contínuo funcionamento do sistema, executando ações no sentido de minorar as consequências dessas falhas, ou até mesmo recuperar completamente. Kephart e Chess (2003) definiram a capacidade de Auto-Recuperação como o mecanismo através do qual o Sistema Autónomo recupera de falhas ocorridas, e pelo qual prevê possíveis desvios aos objetivos pretendidos, podendo a partir destas previsões proceder às devidas alterações.

5.2.4. Auto-Proteção

Os Sistemas Autónomos devem estar dotados de capacidades de combate a ataques maliciosos ou falhas em cascata, sendo capazes de identificar, prever e tentar evitar ou remediar situações anormais. A função da Auto-Proteção consiste em defender o sistema em dois sentidos (Kephart e Chess, 2003):

- Deve protegê-lo a larga escala contra ataques maliciosos e intrusivos tais como vírus, acessos não autorizados e ataques *DoS* (*denial-of-service*);
- Deve ser capaz de, com base em relatórios de sensores, identificar problemas que possam vir a ocorrer para que estes possam ser evitados ou atenuados. Estes problemas podem ser comportamentos errados que provocam desvios ao objetivo pretendido. Devem ser tomadas decisões baseadas em políticas para corrigir o sistema tornando-o menos vulnerável.

5.3. Arquitetura

Um sistema autónomo é constituído por vários elementos que colaboram entre si de modo a proporcionar todas as características de que necessita. De uma forma genérica, um sistema autónomo pode ser dividido em vários componentes, ilustrados na Figura 5.1 (IBM, 2005; IBM, 2006; Kephart e Chess, 2003; Parashar e Hariri, 2006). Estes componentes encontram-se interligados e comunicam através de serviços de comunicação.

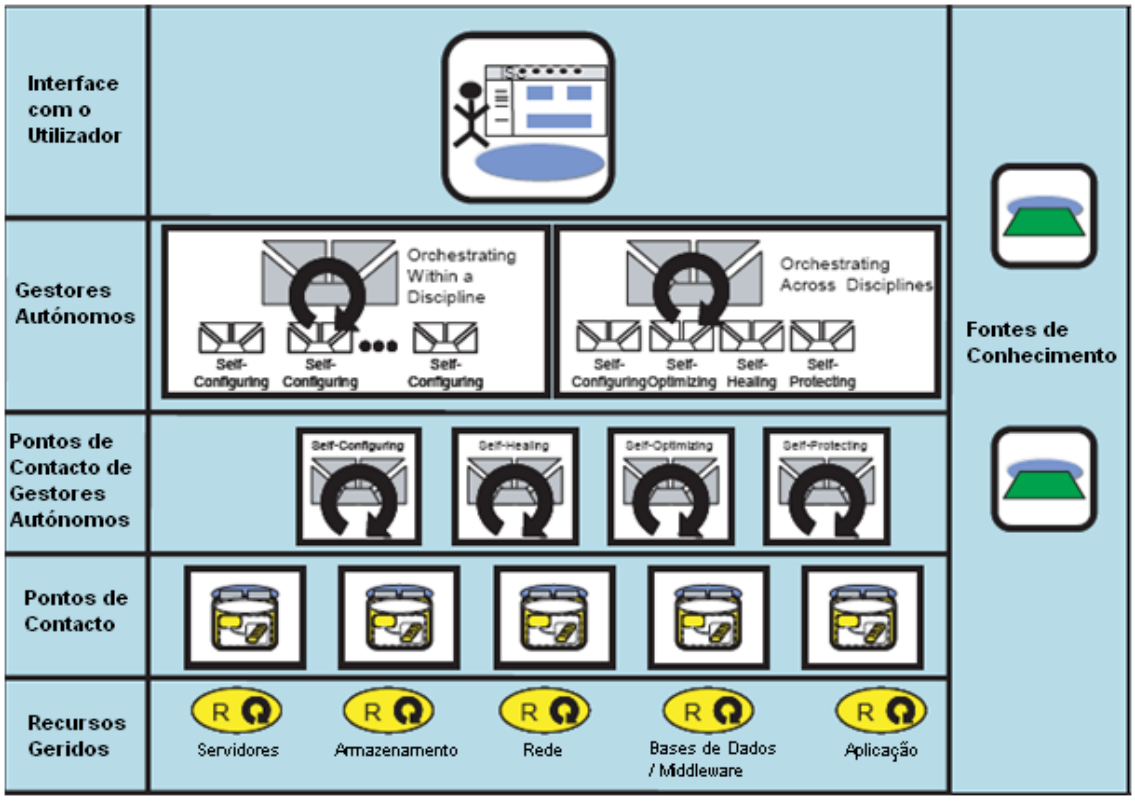

Figura 5.1 – Arquitetura de um sistema de Computação Autónoma (adaptado de (IBM, 2005))

5.3.1. Fontes de conhecimento

Central a todas as atividades de um Sistema Autónomo considera-se uma base de conhecimento que é caracterizada por diferentes funcionalidades perante os requisitos de aplicações específicas.

A fonte ou base de conhecimento refere-se a qualquer tipo de registos de informação que possibilite armazenar conhecimento. Consiste em tipos particulares de dados com sintaxe e semântica tais como sintomas, políticas, alteração de planos ou pedidos, de forma a poderem ser partilhados pelos gestores autónomos do sistema (Kephart e Chess, 2003).

À medida que o sistema se vai autodesenvolvendo, este aprende a detetar novos tipos de comportamento, e inclui políticas introduzidas pelos gestores. Essas informações são armazenadas nas fontes de conhecimento sob a forma de topologias, métricas, sintomas, políticas, e ficheiros de *log*. Assim, os gestores autónomos podem aceder ao conhecimento para a tomada de decisão de três formas diferentes:

- **A informação é enviada**: uma política é enviada e consiste numa coleção de comportamentos limitados ou preferenciais que influenciam as decisões que o gestor autónomo tem que tomar;
- **O conhecimento é enviado a partir de uma fonte de conhecimento externa**: o gestor autónomo pode obter definições de sintomas ou conhecimento de recursos específicos;
- **O gestor autónomo gera automaticamente o seu conhecimento**: o conhecimento pode ser criado pela monitorização, baseado na informação recolhida dos sensores. A execução de um gestor autónomo pode atualizar o conhecimento para registar as ações que foram realizadas como resultado da análise e planeamento. O gestor autónomo pode então indicar que efeitos tiveram as ações efetuadas no recurso gerido, após efetuar a respetiva monitorização. Esse conhecimento pode ser armazenado na fonte de conhecimento, podendo ser partilhado por outros gestores autónomos, ou fica apenas contido no próprio gestor.

Em sistemas autónomos existem vários tipos de informação, armazenados segundo sintaxe e semântica comum, de forma que possam ser partilhados por diversos componentes (Kephart e Chess, 2003):

- **Topologias de soluções:** corresponde a conhecimento acerca da configuração e construção dos componentes do sistema. O plano de funcionamento de um Sistema Autónomo baseia-se neste tipo de conhecimento para o seu plano de instalação e configuração;
- **Políticas**: conceito crucial na Computação Autónoma, refere-se a um tipo de conhecimento que é consultado para se determinar se é ou não necessário efetuar ações no sistema. Todos os gestores autónomos podem ter acesso a elas e, dessa forma, todo o sistema é gerido através de uma dada coleção de políticas. O sistema deve igualmente fornecer interfaces para a criação, alteração e implementação de políticas, e mesmo saber distinguir se as regras estão especificadas corretamente, protegendo-se de erros humanos;

- **Deteção de problemas**: são incluídos dados monitorizados, árvores de decisão, e conhecimento criado pelo processo de deteção de erros. Como o sistema reage de forma a corrigir problemas detetados, as informações relativas a essas correções são armazenadas como conhecimento.

5.3.2. Interface com o utilizador

A interface com o utilizador promove uma apresentação consistente aos utilizadores permitindo-lhes, de uma perspetiva de administração, inserir determinados tipos de informações necessárias, desde políticas até objetivos para o Sistema Autónomo. Pretende-se que funcione como plataforma de controlo do sistema, onde seja possível controlar todos os componentes. Pode ser possível colaborar com outros Gestores Autónomos ao mesmo nível ou controlar gestores de níveis mais baixos. A interface com o utilizador possui vantagens a nível de custos na organização, pois, para além de aumentar a eficácia da administração, reduz a necessidade dos utilizadores receberem formação para lidarem com os novos componentes introduzidos no sistema.

Em alguns casos, um administrador pode escolher requerer intervenção humana para determinadas tarefas, e a interação humana com o sistema pode ser melhorada com uma interface gráfica apropriada, construída com base em padrões industriais. O principal objetivo dessa interface gráfica é fornecer uma plataforma única onde podem estar todas as funções administrativas de produtos de *software*, de servidores e de armazenamento, de modo a permitir ao utilizador gerir as soluções em vez de gerir componentes ou produtos. As funções administrativas variam desde configuração da solução até controlo e monitorização em tempo-real (IBM, 2005).

No que diz respeito a uma interface integradora de soluções, o cliente valoriza o reduzido custo de aquisição e curvas de aprendizagem sempre que novos produtos ou soluções são adicionados ao ambiente autónomo. A redução das curvas de aprendizagem é conseguida através do uso de padrões. Ao fornecer formatos de apresentação e comportamentos consistentes para funções de administração entre diversos produtos, a interface integradora torna-se familiar ao utilizador, eliminando a necessidade dos utilizadores aprenderam uma nova interface sempre que um novo produto é introduzido (IBM, 2005).

5.3.3. Gestores autónomos

Um Sistema Autónomo é composto por um ou mais gestores autónomos, tendo cada um a função de monitorizar continuamente o sistema e tratar eventos que necessitem de tratamento. Para isso, faz uso do ponto de contacto, com os sensores a permitirem ler o estado e com os atuadores a permitirem modificá-lo.

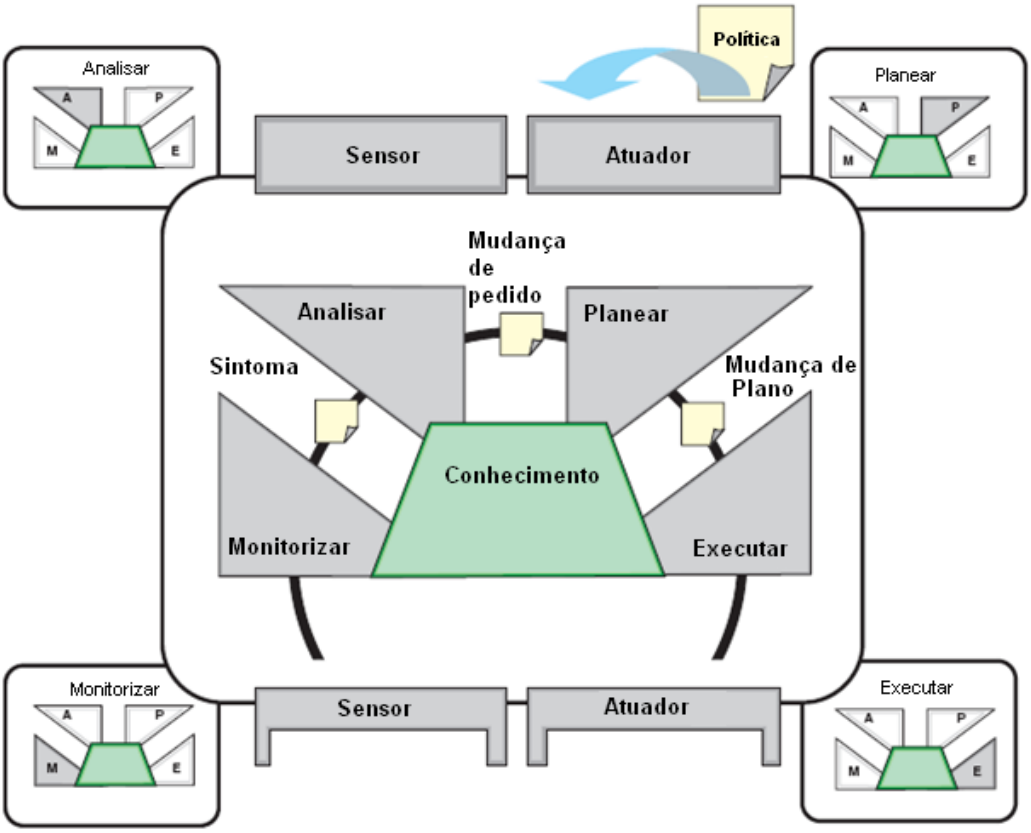

Figura 5.2 – Funcionamento de um gestor autónomo (adaptado de (IBM, 2005))

O ciclo de controlo de um gestor autónomo ilustrado na Figura 5.2, consiste num ciclo fechado composto pelas seguintes fases (Kephart e Chess, 2003):

- **Monitorização**: fornece os mecanismos para recolher, agrupar, filtrar e reportar detalhes acerca de um dado recurso gerido. O gestor autónomo deve monitorizar constantemente o sistema através dos sensores, cuja informação obtida é convertida em sintomas que podem ser analisados;
- **Análise**: fornece os mecanismos que suportam o gestor autónomo a aprender sobre o sistema e a prever situações futuras semelhantes. A análise é efetuada

com base na informação recolhida pela monitorização e a interpretação e avaliação com base no conhecimento do sistema;

- **Planeamento**: fornece os mecanismos necessários para a construção planos de ação, mediante políticas armazenadas, que permitem remediar situações analisadas anteriormente;
- **Execução**: fornece os mecanismos que executam as ações indicadas pelo planeamento. Os planos devem ser colocados em execução com o mínimo de interferência e de conflitos entre si. As alterações são efetuadas através dos atuadores dos recursos geridos que necessitem de ser atualizados.

Podem ser realizadas algumas adaptações úteis à sua implementação. Pode ser configurado para efetuar apenas determinadas partes do ciclo de controlo, ficando deste modo dedicado a essa finalidade. Outra configuração possível é integrar dois Gestores Autónomos que estejam configurados com funções de monitorização e análise, e planeamento e execução respetivamente, de forma a criar um ciclo fechado mais completo.

5.3.4. Pontos de contacto

Nos Sistemas Autónomos, os pontos de contacto representam a interface de gestão para os recursos geridos (servidores, unidades de armazenamento, etc.). Um ponto de contacto ativa estes recursos para serem geridos independentemente do tipo de recurso. Os recursos geridos são então acedidos e controlados pela interface de gestão que fornece mecanismos como ficheiros de *log*, eventos, comandos, aplicações, etc.

Esta interface de gestão possui dois mecanismos para interação e controlo dos recursos: o sensor e o atuador. O sensor representa um conjunto de propriedades que expõem informação acerca do estado corrente de um recurso gerido, e que acedem através de operações "*get*". O atuador, por sua vez, representa um conjunto de operações "*set*", que de alguma forma permitem alterar o estado de um recurso gerido.

5.3.5. Recursos geridos

Os recursos geridos englobam todo o tipo de recursos de *hardware* e *software* que os componentes do sistema controlam, e podem ser servidores, unidades de armazenamento, serviços, ou outras entidades. Um recurso gerido pode conter embutido o seu próprio ciclo de controlo de autogestão, embora este ciclo possa não ser externamente

visível. Quando os detalhes do ciclo de controlo se encontram visíveis, este é gerido através da interface de gestão fornecida para o recurso gerido.

5.3.6. Serviços de comunicação

Os serviços de comunicação, ou *Enterprise Service Bus* (Draper e Sweitzer, 2006; Ganek, 2006), auxiliam a interligação e a comunicação dos componentes de um Sistema Autónomo. A função destes serviços é realizada segundo os seguintes padrões:

- Agrupar múltiplos mecanismos de gestão para um recurso gerido;
- Ativar um gestor autónomo para controlar múltiplos pontos de contacto;
- Ativar um gestor autónomo para controlar um único ponto de contacto;
- Ativar múltiplos gestores autónomos para controlarem múltiplos pontos de contacto.

Draper e Sweitzer (2006) utilizaram os serviços de comunicação para resolver uma série de situações tais como evitar o aumento de complexidade devido ao acréscimo do número de gestores autónomos e elementos a gerir, e como agregar gestores autónomos parciais para formar outros mais completos.

5.4. Desafios da Computação Autónoma

Nami e Sharifi (2007) referiram alguns desafios presentes na Computação Autónoma, desde relações entre elementos autónomos, passando por aprendizagem e otimização, até à robustez e confiança. Os relacionamentos entre elementos autónomos apresentam um papel fundamental na implementação das capacidades de autogestão. Estas relações têm um ciclo de vida composto por especificação, localização, negociação, disposição, aprovisionamento, operação, e término. Cada uma destas fases apresenta os seus próprios desafios (Kephart e Chess, 2003). Expressar o conjunto de serviços de saída que um elemento autónomo pode executar e o conjunto de serviços de entrada que este requer num formulário padrão, bem como estabelecer a sintaxe e a semântica dos serviços padrão, pode ser um desafio na sua especificação. Como um elemento autónomo deve dinamicamente localizar os serviços de entrada que necessita, também os outros elementos que precisam dos serviços de saída devem dinamicamente localizar este elemento, o que faz com que a confiança nos elementos autónomos seja uma área promissora de investigação. Os elementos autónomos requerem também protocolos e estratégias de forma

a estabelecer regras de negociação e gerir o fluxo de mensagens entre os negociadores. Um dos desafios passa por desenvolver e analisar algoritmos e protocolos de negociação, determinando que algoritmos e protocolos podem ser mais eficazes. O aprovisionamento autónomo pode também ser uma área de investigação, pois após chegarem a acordo, os gestores autónomos controlam a situação e se o acordo é violado, várias soluções podem ser introduzidas.

A principal ideia de transferir o conhecimento do sistema de gestão dos peritos humanos para os sistemas autónomos passa por observar a forma como os peritos humanos resolvem um problema em vários sistemas e, através do rastro das suas atividades, é possível criar um processo de aprendizagem robusto (Bigus *et al.*, 2002). Este processo pode executar automaticamente a mesma tarefa num novo sistema. Esta aquisição de conhecimento dos peritos humanos e a produção de sistemas que incluem este conhecimento apresenta ainda interessantes desafios para a sua implementação.

Uma das razões para o sucesso dos sistemas autónomos é a capacidade de se gerirem autonomamente e reagir às mudanças ou perturbações. Em sistemas autónomos sofisticados, os componentes individuais que interagem uns com os outros devem ser capazes de se adaptarem a ambientes dinâmicos e aprender a resolver problemas com base na experiência. Assim, a otimização apresenta espaço e oportunidade de contribuições, pois, nestes sistemas, a adaptação autónoma altera o comportamento dos agentes.

Um dos principais desafios na Computação Autónoma passa pela robustez dos sistemas. O conceito de robustez tem muitos significados, uma vez que tem sido usada em várias áreas desde a ecologia, a engenharia, e os sistemas sociais. É possível interpretar a robustez como estabilidade, confiança, capacidade de sobrevivência, e tolerância a falhas, embora não signifique todos estes conceitos. Refere-se a robustez como a capacidade de um sistema manter as suas funções num estado ativo, e persistir quando ocorrem alterações na estrutura interna do sistema ou do ambiente externo, e é muitas vezes confundida com estabilidade. Embora tanto a estabilidade como a robustez tenha o seu foco na persistência, a robustez é mais ampla do que a estabilidade (Jen, 2003). É possível que os componentes de um sistema não sejam robustos, mas as interconexões entre eles tragam robustez ao nível do sistema. Um sistema robusto pode executar múltiplas funcionalidades para resistir, sem alterar a sua estrutura. Com a especificação de instruções que permitem aos sistemas preservar a sua identidade mesmo quando estes são interrompidos, a robustez dos sistemas pode aumentar.

Outro desafio identificado passa pela questão da confiança, que pode ser discutida de duas formas: a confiança entre elementos autónomos e a confiança entre sistemas autónomos e o seu utilizador. A confiança entre elementos autónomos pode ser uma questão importante no seu relacionamento e afeta o funcionamento do sistema autónomo. Tal significa que um elemento autónomo espera que outro elemento seja confiável a ponto de realizar as suas tarefas para fornecer os serviços acordados de modo a atingir um determinado objetivo. Num elemento autónomo, a confiança pode ser construída de acordo com a análise do histórico do desempenho desse elemento. A confiança pode ser definida por um agregado de atributos tal como a segurança e a competência. Neste caso, cada um dos atributos precisa de ser mensurado. Uma das questões mais importantes está relacionada com a forma como essa medição é automaticamente efetuada, e como as ferramentas usadas podem melhorar o desempenho e confiança dos Sistemas Autónomos.

A confiança entre um sistema autónomo e os seus utilizadores é um fator essencial de aceitação desse sistema por parte dos utilizadores. No nível mais alto de confiança, cada execução do sistema autónomo é aceite sem intervenção do utilizador. Para alcançar este objetivo, cada execução dos gestores autónomos é examinada através da atribuição de um índice entre 0 e 1 a cada gestor autónomo e através da atribuição do mesmo dígito a um grupo de gestores autónomos baseada no desempenho a longo prazo de cada gestor ou grupo de gestores autónomos (Chan *et al.*, 2005). Um dígito perto de 1 indica um nível alto de confiança.

5.5. Áreas de aplicação

Desde a sua proposta, a Computação Autónoma tem sido usada em vários domínios, desde a *Grid Computing*, aos sistemas de gestão de energia, até à computação ubíqua (Huebscher e McCann, 2008).

Uma área onde a Computação Autónoma pode contribuir de forma significativa é na ***Grid Computing***, uma vez que estes ambientes são inerentemente grandes, heterogéneos e dinâmicos, originando desafios no seu desenvolvimento, configuração e gestão (Parashar, 2006). Exemplos de aplicação da Computação Autónoma em *Grid Computing* podem ser encontrados em (Bahrami *et al.*, 2010; Bangerth *et al.*, 2005; Chopra e Singh, 2011; Matossian *et al.*, 2005; Xiangliang *et al.*, 2010).

Outro domínio de aplicação da Computação Autónoma é a área dos **sistemas de gestão de energia**. Estima-se que os equipamentos de energia, arrefecimento e

eletricidade são responsáveis por 63% do custo total da estrutura física de um centro de dados. Estas estatísticas motivaram a investigação em sistemas autoadaptativos que não só otimizam a gestão de recursos em termos de desempenho mas, também, em termos de energia consumida por um determinado algoritmo ou serviço numa determinada infraestrutura (Huebscher e McCann, 2008). Exemplos de trabalhos nesta área são (Das *et al.*, 2008; Femal e Freeh, 2005; Kandasamy *et al.*, 2004; Khargharia *et al.*, 2010; Khargharia *et al.*, 2006).

A Computação Autónoma tem também vindo a ser usada em **centros de processamento de dados**, conjuntos de computadores heterogeneamente distribuídos em áreas amplas com o objetivo de executar desde aplicações científicas a aplicações comerciais, por parte de diferentes utilizadores. Dada a distribuição geográfica, a manutenção deste tipo de sistemas traz uma complexidade acrescida (Huebscher e McCann, 2008). Neste domínio, a Computação Autónoma foi usada em, e.g. (Kaminsky *et al.*, 2008; Khargharia *et al.*, 2010; Khargharia *et al.*, 2006; Rodero *et al.*, 2010).

A **computação ubíqua** e a Computação Autónoma convergiram num ponto em que a tecnologia dispersa nos ambientes, juntamente com a capacidade das pessoas interagirem entre si, permite tirar partido do seu uso para novas potencialidades. A computação ubíqua conjetura ambientes repletos de dispositivos, aplicações e serviços que possibilitam uma união do mundo real com o mundo virtual, mas de uma forma natural. A promessa da criação desses ambientes não se tornará possível sem que a sua complexidade desapareça da perceção dos utilizadores. Para que tal seja possível, estes sistemas necessitam de autonomia, de forma a gerir a sua própria configuração e evolução com o mínimo de intervenção do utilizador. É neste contexto que surge a noção de sistemas ubíquos autónomos que envolvem as facetas de autoconfiguração e autorreparação (Machado, 2009). Exemplos de aplicação da computação ubíqua autónoma podem ser encontrados em (Hassan *et al.*, 2009; McCann e Sterritt, 2010; Peizhao *et al.*, 2008; Ranganathan, 2004; Ranganathan *et al.*, 2005).

Além dos domínios mais comuns, a Computação Autónoma tem também sido aplicada em **escalonamento**, tal como acontece no âmbito desta tese. Várias aplicações neste domínio podem ser encontradas, e.g. (Madureira *et al.*, 2009a; Pereira, 2009; Pereira e Madureira, 2013; Pereira *et al.*, 2013a; Ramirez *et al.*, 2008; Ross e Bambos, 2006; Whiteson e Stone, 2004).

5.6. Sumário

Neste capítulo introduziu-se o conceito de Computação Autónoma, descrevendo-se os comportamentos intrínsecos de Auto-Configuração, Auto-Otimização, Auto-Recuperação, e Auto-Proteção. Foram também apresentados aspetos da arquitetura dos sistemas autónomos bem como alguns desafios e áreas de aplicação.

Capítulo 6. Aprendizagem em Sistemas Multiagente e na afinação de parâmetros

6.1. Introdução

Os Sistemas Multiagente atuam tipicamente em ambientes dinâmicos, complexos, abertos e imprevisíveis. Assim, a necessidade destes sistemas aprenderem torna-se, por vezes, inevitável. Plaza *et al.* (1997) definiram a aprendizagem como um processo de *"improvisar o desempenho individual, precisão (ou qualidade) de soluções, eficiência (ou rapidez) de pesquisa soluções e âmbito de problemas resolúveis"*. Embora esta definição seja bastante útil, revela uma visão limitada e rígida do conceito de aprendizagem. De uma forma mais geral, pode-se considerar a capacidade de aprendizagem como a aquisição de novo conhecimento ou a atualização do conhecimento existente (Pereira, 2009).

A Aprendizagem Automática é uma área de investigação da Inteligência Artificial que se dedica ao desenvolvimento de técnicas e algoritmos para dotar os sistemas da capacidade de aprendizagem, requerendo o mínimo de intervenção humana. Mitchell (1997) definiu que *"a área da Aprendizagem Automática se preocupa com a questão de como construir programas computacionais que melhorem automaticamente com a experiência"*. Por seu lado, Alpaydin (2004) refere que *"a Aprendizagem Automática consiste em programar os computadores de forma a otimizar um critério de desempenho através de exemplos ou experiência passada"*. Em ambos os estudos é notória a referência ao mesmo conceito, que se baseia no termo **experiência**. A experiência pode definir-se como o "ato ou efeito de experimentar", mas também como o "conhecimento obtido pela prática de uma atividade ou pela vivência" [2]. É assim possível concluir da necessidade de haver um período de existência para experimentar, de modo a ser possível adquirir a experiência com a qual é possível desenvolver a capacidade de aprendizagem, sendo válido para os humanos, animais ou mesmo computadores/máquinas (Pereira, 2009).

A Aprendizagem Automática preocupa-se com as mesmas questões da Psicologia, de Estatística, e de *Data Mining*, mas com ênfase diferente. A Psicologia acerca da aprendizagem humana aspira compreender os mecanismos subjacentes aos vários comportamentos de aprendizagem exibidos pelas pessoas (aprendizagem conceptual, aquisição de habilidades, mudança de estratégias, etc.). A Estatística foca-se em

[2] http://www.infopedia.pt

compreender os fenómenos que geram os dados, muitas vezes com o objetivo de testar diferentes hipóteses sobre esses fenómenos. Em *Data Mining* procura-se encontrar padrões nos dados. No entanto, a Aprendizagem Automática preocupa-se principalmente com o desempenho do sistema computacional resultante. As áreas de aplicação de Aprendizagem Automática são várias, onde se inclui o processamento de linguagem natural, o reconhecimento de padrões, a análise de mercados, a classificação de sequências de ADN, o reconhecimento de voz e escrita, o reconhecimento de objetos em visão por computador, os jogos virtuais, a locomoção robótica, entre outros (Alpaydin, 2004; Sammut e Webb, 2011).

Neste capítulo, discute-se a relação entre Aprendizagem Automática e Sistemas Multiagente, identificam-se algumas técnicas de Aprendizagem Automática (secção 6.2), e referem-se alguns aspetos de aprendizagem em agentes (secção 6.3). Na secção 6.4 são apresentadas algumas técnicas de aprendizagem aplicadas ao problema de afinação de parâmetros, com especial foco nos algoritmos de *Racing* e no Raciocínio baseado em Casos.

6.2. Técnicas de Aprendizagem Automática

Os humanos, e os animais em geral, reagem ao seu ambiente e realizam diferentes ações em resposta a estímulos que recebem. Considera-se, geralmente, um sinal de inteligência se um determinado sujeito responde às alterações que ocorrem ao longo do tempo e se tornam mais eficazes. Quando isto acontece, diz-se que o sujeito é capaz de aprender (Birattari, 2009). A aprendizagem é um fator fundamental da inteligência humana assim como na Inteligência Artificial. A Aprendizagem Automática estuda algoritmos de aprendizagem, os quais especificam como as alterações no comportamento do aprendiz depende dos estímulos recebidos e no *feedback* do ambiente. Dependendo deste *feedback*, é possível identificar claramente três categorias de Aprendizagem Automática, que se distinguem pelo tipo de informação tratada, tal como referido na literatura (Alpaydin, 2004; Birattari, 2009; Mitchell, 1997; Sammut e Webb, 2011): **Aprendizagem Supervisionada** (emprega dados rotulados), **Aprendizagem Não-Supervisionada** (utiliza dados não rotulados) e **Aprendizagem por Reforço** (com o objetivo de maximizar uma recompensa).

A Aprendizagem Supervisionada preocupa-se em criar uma função, ou regra de decisão, para efetuar uma previsão/decisão sobre novos dados, a partir de um conjunto de dados de treino. Estes dados de treino consistem em pares de objetos de entrada e

respetivos dados de saída, tendo cada objeto a sua classe (rótulo) própria. A função criada pode gerar um valor contínuo (regressão) ou pode prever a classe do novo objeto (classificação). A tarefa do algoritmo supervisionado é prever o valor da função para qualquer objeto de entrada válido depois de analisar um conjunto de exemplos de treino. *Artificial Neural Networks* (Bishop, 1995), *Decision Tree Learning* (Breiman *et al.*, 1984), *Random Forests* (Breiman, 2001), e *Support Vector Machines* (Cortes e Vapnik, 1995) são exemplos de técnicas de Aprendizagem Supervisionada.

A Aprendizagem Não-Supervisionada apresenta como objetivo determinar como os dados estão organizados. Contrariamente ao que acontece com a Aprendizagem Supervisionada, pretende-se aprender sem recorrer a exemplos rotulados. Nestes dados de entrada não-rotulados, certos padrões ocorrem mais do que outros e pretende-se verificar o que acontece (ou não) geralmente. Exemplos de técnicas de Aprendizagem Não-Supervisionada são: *K-Means Clustering* (Hartigan, 1975), Hidden Markov Models (Rabiner e Juang, 1986), e Self-Organizing Maps.

Por vezes, a literatura refere uma categoria adicional, situada entre a Aprendizagem Supervisionada e a Aprendizagem Não-Supervisionada, designada de **Aprendizagem Semi-Supervisionada**, que usa tanto dados rotulados como não-rotulados. Tipicamente, utiliza-se uma pequena quantidade de dados rotulados juntamente com uma grande quantidade de dados não-rotulados. Os dados não-rotulados, quando usados juntamente com pequenas quantidades de dados rotulados, podem produzir melhorias significativas na precisão do processo de aprendizagem. Co-training (Blum e Mitchell, 1998), *Expectation Maximization* (Dempster *et al.*, 1977), e *Transductive Support Vector Machines* (Vapnik, 1998) são exemplos de técnicas de Aprendizagem Semi-Supervisionada.

A Aprendizagem por Reforço (*Reinforcement Learning*) é a área da Aprendizagem Automática que se preocupa em como um agente deve tomar ações num ambiente para maximizar uma determinada recompensa (ou reforço). É bastante utilizada devido à sua simplicidade, flexibilidade e eficácia em muitos domínios de problemas. Sutton e Barto (1998) definiram a Aprendizagem por Reforço como a *"aprendizagem de uma política ótima de mapeamento de situações para ações de forma a maximizar um sinal numérico de recompensa"*. A Aprendizagem por Reforço distingue-se da Aprendizagem Supervisionada no facto de nunca ter pares de entrada/saída corretos, nem ações sub-ótimas explicitamente corretas. Além disso, existe um foco no desempenho, o que torna necessário que se encontre um equilíbrio entre exploração e aproveitamento. Os métodos de Aprendizagem por Reforço são particularmente úteis em domínios onde a informação reforçada (expressa

por penalizações ou recompensas) é fornecida após uma sequência de ações serem executadas no ambiente (Panait e Luke, 2005). Exemplos de técnicas de Aprendizagem por Reforço são: *Q-Learning* (Watkins e Dayan, 1992) e *Temporal Difference Learning* (Sutton e Barto, 1998).

Na literatura é também bastante comum a referência a uma outra categoria, conhecida por Aprendizagem baseada em Instâncias[3] (Mitchell, 1997) ou por Métodos Não-Paramétricos (Alpaydin, 2004), onde se inclui o **Raciocínio baseado em Casos**, descrito na secção 6.4.2 uma vez que se revelou bastante importante no desenvolvimento deste trabalho.

6.3. Aprendizagem em Sistemas Multiagente

Num Sistema Multiagente torna-se difícil, ou até mesmo impossível, determinar corretamente o comportamento das suas atividades concretas *a priori*. Para isso seria necessário, por exemplo, saber quais os requisitos do ambiente que irão surgir no futuro, quais os agentes que estarão disponíveis na altura, como é que esses agentes poderão ter de interagir em resposta aos requisitos, etc. Tais problemas, resultantes da complexidade dos Sistemas Multiagente, podem ser evitados, ou pelo menos reduzidos, dotando os agentes inteligentes da capacidade de melhorar os desempenhos futuros do sistema global, ou de parte do sistema.

Tal como referido no Capítulo 4**,** a inteligência de um agente implica um certo grau de autonomia, o que requer a capacidade de tomar decisões de forma autónoma. Na maioria dos ambientes dinâmicos não é possível prever todas as situações que um agente pode encontrar, sendo necessário que o agente tenha a capacidade de se adaptar a novas situações. Isto verifica-se especialmente nos Sistemas Multiagente, onde em muitos casos o comportamento global emerge, em vez de ser pré-definido. Consequentemente, a aprendizagem é um componente crucial da autonomia e tem sido alvo de estudo por parte da comunidade de Sistemas Multiagente (Alonso *et al.*, 2001; Panait e Luke, 2005).

De acordo com Goldman e Rosenschein (1996), a capacidade de aprendizagem em ambientes multiagente pode ajudar os agentes a melhorar o seu desempenho. Agentes que atuem em ambientes dinâmicos podem reagir a eventos inesperados, através da generalização do que aprenderam na fase de treino. Nos sistemas cooperativos de

[3] Ou Aprendizagem baseada em Memória, é uma família de métodos de Aprendizagem Automática que armazena os exemplos de treino e constrói hipóteses diretamente a partir dessas instâncias de treino.

resolução de problemas, o comportamento cooperativo pode ser concretizado mais eficientemente quando os agentes se adaptam à informação sobre o ambiente e sobre os seus parceiros. Agentes que aprendem uns com os outros podem, por vezes, evitar a coordenação repetida das suas ações a partir do zero, para problemas semelhantes. Podem também ser capazes de, em algumas situações, evitar comunicações em tempo de execução recorrendo a conceitos de coordenação já aprendidos, o que é especialmente útil quando os agentes não têm tempo suficiente para negociar.

Alguns Sistemas Multiagente com capacidade de aprendizagem são implementados com agentes de treino (agentes tutores), os quais, neste caso, se tornam nas principais fontes de conhecimento de aprendizagem do sistema.

A Aprendizagem Automática explora meios para automatizar o processo indutivo, por exemplo, colocar um agente-máquina a descobrir por si próprio, através de várias tentativas, como resolver uma dada tarefa ou para minimizar o erro (Panait e Luke, 2005). Embora tenha vindo a ser estudada extensivamente ao longo dos anos, a Aprendizagem Automática tem sido na sua maioria investigada separadamente e só recentemente tem recebido maior atenção por parte da área dos Sistemas Multiagente (Alonso *et al.*, 2001).

Alguns Sistemas Multiagente apresentam o problema da aprendizagem distribuída, isto é, muitos agentes a aprenderem separadamente para atingir um resultado global comum. Os algoritmos de aprendizagem referidos na literatura têm vindo a ser desenvolvidos para aprendizagem de agente único de hipóteses separadas e independentes. Uma vez que o processo de aprendizagem é distribuído por vários agentes aprendizes, os algoritmos de aprendizagem existentes necessitam de modificações, ou então é necessário que sejam desenvolvidos novos algoritmos. Na aprendizagem distribuída, os agentes precisam de cooperar e comunicar para aprenderem de forma efetiva (Alonso *et al.*, 2001).

Panait e Luke (2005) focaram-se na aplicação da Aprendizagem Automática em Sistemas Multiagente. Segundo estes autores, a Aprendizagem Automática mostrou ser uma abordagem popular para a resolução de problemas em Sistemas Multiagente considerando a complexidade inerente a muitos desses problemas, o que conduz a soluções proibitivamente complexas. Focaram especificamente os domínios de problemas onde os múltiplos agentes cooperam para resolver uma tarefa, designada **Aprendizagem Cooperativa Multiagente** (Panait e Luke, 2005).

Devido à complexidade inerente nas interações entre os múltiplos agentes, os métodos de Aprendizagem Automática (nomeadamente os métodos de Aprendizagem Supervisionada) não são facilmente aplicáveis ao problema porque assumem tipicamente a existência de um "perito" que possa fornecer aos agentes o comportamento "correto" para uma determinada situação. No entanto, podem ser encontradas algumas exceções em (Garland e Alterman, 2004; Goldman e Rosenschein, 1996; Williams, 2004). Assim, têm vindo a ser usados maioritariamente métodos de Aprendizagem por Reforço, que apresentam provas teóricas de convergência mas, infelizmente, algumas dessas suposições de convergência não se aplicam a muitas aplicações de mundo real, incluindo os Sistemas Multiagente.

De acordo com Stone e Veloso (1998), a Aprendizagem de Agente Único foca-se em como um agente melhora as suas capacidades individuais, independentemente do domínio onde está inserido. Normalmente, este tipo de aprendizagem consiste na aplicação de métodos de Aprendizagem Automática e é a base para a Aprendizagem Multiagente. Por um lado, a Aprendizagem de Agente Único pode nem sempre levar a um desempenho ótimo em ambientes multiagente e podem existir domínios em que a Aprendizagem Multiagente é mais natural e mais eficaz. No entanto, não é possível dissertar acerca de Aprendizagem Multiagente se o que um agente aprende não afeta ou não é afetado pelos outros agentes vizinhos (Alonso *et al.*, 2001).

Define-se genericamente a Aprendizagem Multiagente como a aplicação da Aprendizagem Automática a problemas que envolvem múltiplos agentes. Panait e Luke (2005) referem duas características da Aprendizagem Multiagente que requerem o seu estudo fora do âmbito da Aprendizagem Automática. Primeiro, como a Aprendizagem Multiagente lida com os domínios de problemas que envolvem múltiplos agentes, o espaço de procura pode ser invulgarmente grande, e, devido à interação entre esses agentes, pequenas mudanças nos comportamentos aprendidos podem resultar muitas vezes em alterações imprevisíveis nas propriedades do Sistema Multiagente como um todo. Segundo, a Aprendizagem Multiagente pode envolver muitos aprendizes, cada um a aprender individualmente e a adaptar-se no contexto dos outros.

Os algoritmos de Aprendizagem Multiagente encontrados na literatura são, na sua maioria, baseados em algoritmos de Aprendizagem de Agente Único. A distinção reside na existência de outros agentes e interações na Aprendizagem Multiagente, pois um agente pode aprender conhecimento a partir de outros agentes bem como a partir do ambiente. Os objetivos da Aprendizagem Multiagente podem ser a coordenação, a aprendizagem dos

modelos e intenções dos outros agentes, a comunicação efetiva com outros agentes, entre outros, sendo que nenhum destes objetivos existe na Aprendizagem de Agente Único. No entanto, do ponto de vista algorítmico, a Aprendizagem Multiagente não parece ser muito diferente da Aprendizagem de Agente Único (Panait e Luke, 2005).

A hierarquia pode ser o fator mais importante durante o desenvolvimento da Aprendizagem Multiagente. A hierarquia de aprendizagem pode ser considerada a dois níveis: a hierarquia de algoritmos de aprendizagem e a hierarquia organizacional de agentes. A Aprendizagem Multiagente com apenas um método de aprendizagem, tal como Aprendizagem por Reforço ou Redes Neuronais, não é suficiente para sistemas complexos. Para lidar com estes problemas, a hierarquia de algoritmos de aprendizagem é igualmente importante. A Aprendizagem em Camadas (Stone, 2000) é um bom exemplo, e combina um organizador de alto-nível e um módulo de aprendizagem de baixo-nível. A hierarquia organizacional pode ser predefinida, aprendida ou pode emergir das ações dos agentes.

É também possível analisar a Aprendizagem Multiagente numa perspetiva cooperativa, recaindo na área da Aprendizagem Cooperativa Multiagente (Panait e Luke, 2005), é possível definir duas categorias principais de abordagens de Aprendizagem Cooperativa Multiagente. A primeira, **Aprendizagem para a Equipa** (subsecção **Erro! A origem da referência não foi encontrada.**), aplica um único aprendiz para procurar comportamentos para a comunidade de agentes. A segunda categoria de técnicas, **Aprendizagem Concorrente** (subsecção 6.3.2), utiliza múltiplos processos de aprendizagem concorrente, e, embora a sua designação pareça antagónica relativamente à Aprendizagem Cooperativa, o seu objetivo passa igualmente pela cooperação e melhoria do desempenho global do Sistema Multiagente. Em vez de aprenderem comportamentos para a comunidade inteira, as abordagens de Aprendizagem Concorrente empregam um aprendiz por cada membro da equipa, na esperança de reduzir o espaço conjunto projetando para N espaços separados. No entanto, a presença de múltiplos aprendizes concorrentes faz com que o ambiente seja não estacionário, o que é uma violação das suposições associadas às técnicas de Aprendizagem Automática tradicionais. Por esta razão, a Aprendizagem Concorrente requer novos (ou versões suficientemente modificadas de) métodos de Aprendizagem Automática.

De forma a ilustrar a diferença entre as duas abordagens, considere-se o cenário de exploração cooperativa descrito no Capítulo 4, onde o objetivo é maximizar o número de itens recolhidos do ambiente. Uma abordagem de Aprendizagem para a Equipa emprega um único aprendiz para, iterativamente, melhorar o “comportamento da equipa” usando a

totalidade de itens recolhidos como medida de desempenho. A abordagem de Aprendizagem Concorrente permite que cada explorador modifique o próprio comportamento a partir dos seus próprios processos de aprendizagem. No entanto, a medida de desempenho deve agora ser repartida entre os vários exploradores (por exemplo, dividindo a recompensa de igual forma por todos os membros da equipa, ou baseada em méritos individuais – por exemplo, quantos itens cada explorador recolheu). Os agentes exploradores melhoram os seus comportamentos independentemente uns dos outros, mas não têm qualquer controlo de sobre o comportamento dos outros agentes.

6.3.1. Aprendizagem para a Equipa

Na Aprendizagem para a Equipa, existe um único aprendiz envolvido, mas com o objetivo de descobrir um conjunto de comportamentos para uma equipa de agentes, em vez de o realizar para um único agente. É uma abordagem simples à Aprendizagem Multiagente porque o aprendiz pode usar técnicas de Aprendizagem Automática de agente único. Este aspeto contorna as dificuldades emergentes da coadaptação de múltiplos aprendizes que se encontram em abordagens de Aprendizagem Concorrente. O aprendiz único preocupa-se com o desempenho da equipa, e não só consigo próprio. Por esta razão, as abordagens de Aprendizagem para a Equipa podem ignorar a atribuição de crédito interagente o que é normalmente difícil de determinar (Panait e Luke, 2005).

A Aprendizagem para a Equipa apresenta também algumas desvantagens (Panait e Luke, 2005). O maior problema refere-se ao elevado espaço de estados para o processo de aprendizagem. Por exemplo, se o agente A pode estar em qualquer um de 100 estados e o agente B pode estar em qualquer um de outros 100 estados, a equipa formada pelos dois agentes pode estar em qualquer um dos 10.000 estados (100x100). Esta explosão do tamanho do espaço de estados pode ser complexa para métodos de aprendizagem que exploram o espaço de utilidades dos estados (tal como a Aprendizagem por Reforço) mas pode não afetar tão drasticamente as técnicas que exploram o espaço de comportamentos (tal como a Computação Evolucionária). Uma outra desvantagem refere-se à centralização do algoritmo de aprendizagem: todos os recursos necessitam de estar disponíveis numa única localização onde o programa irá ser executado. Este aspeto pode ser incómodo em domínios onde os dados estão inerentemente distribuídos.

A Aprendizagem para a Equipa pode ainda ser **homogénea** ou **heterogénea** (Panait e Luke, 2005). Enquanto os aprendizes homogéneos desenvolvem um comportamento único por cada agente, os aprendizes heterogéneos devem lidar com um elevado espaço de

procura, mas com a perspetiva de obter soluções melhores através da especialização dos agentes. Existem ainda abordagens entre as duas categorias que são chamadas de métodos híbridos de Aprendizagem para a Equipa. Na Aprendizagem para a Equipa homogénea todos os agentes apresentam comportamentos idênticos, mesmo que os agentes não sejam idênticos. Na Aprendizagem para a Equipa heterogénea, a equipa é composta por agentes com comportamentos diferentes, com um único aprendiz a tentar melhorar a equipa como um todo. A escolha entre as abordagens depende se são ou não necessários especialistas na equipa. Para informações adicionais sobre Aprendizagem para a Equipa, o leitor pode consultar a referência (Panait e Luke, 2005).

6.3.2. Aprendizagem Concorrente

A alternativa mais comum à Aprendizagem para a Equipa, em Sistemas Multiagente cooperativos, é a Aprendizagem Concorrente, onde múltiplos processos aprendizes tentam melhorar partes da equipa. Tipicamente, cada agente tem o seu próprio processo de aprendizagem para modificar os seus comportamentos. Existem outros níveis de granularidade como, por exemplo, a equipa pode ser dividida em “pelotões”, cada um com o seu próprio aprendiz (Panait e Luke, 2005).

A principal questão na Aprendizagem Concorrente é saber quando é que esta será mais adequada do que a Aprendizagem Homogénea ou Heterogénea em Equipa. Jansen e Wiegand (2003) argumentam que a Aprendizagem Concorrente pode ser preferível nos domínios onde a decomposição é possível e proveitosa, e quando é útil focar cada sub-problema até certo ponto independentemente dos outros. Tal acontece porque a Aprendizagem Concorrente projeta o grande espaço de procura conjunto da equipa em pequenos espaços de procura individuais e separados. Se o problema pode ser decomposto tal que os comportamentos individuais dos agentes sejam relativamente disjuntos, então pode resultar numa dramática redução do espaço de procura e na complexidade computacional. Uma outra vantagem refere-se ao facto de que subdividir o processo de aprendizagem em pedaços mais pequenos permite uma maior flexibilidade no uso dos recursos computacionais para a aprendizagem de cada processo pois estes podem, pelo menos parcialmente, ser aprendidos independentemente uns dos outros.

O principal desafio da Aprendizagem Concorrente consiste em cada aprendiz adaptar os seus comportamentos no contexto dos outros aprendizes sobre os quais não tem controlo. Em cenários de agente único (onde devem ser aplicadas técnicas comuns de Aprendizagem Automática), um aprendiz explora o seu ambiente, e enquanto o faz, melhora

o seu comportamento. Mas existem mudanças com múltiplos aprendizes: conforme os agentes aprendem, modificam os seus comportamentos, o que pode arruinar os comportamentos aprendidos pelos outros agentes (Sandholm e Crites, 1996; Weinberg e Rosenschein, 2004). Uma abordagem simples para lidar com esta coadaptação é tratar os outros aprendizes como parte do ambiente dinâmico para o qual um dado aprendiz se deve adaptar (Schmidhuber, 1996; Schmidhuber e Zhao, 1997). Mas, em Aprendizagem Concorrente, a situação torna-se mais complicada, pois os outros agentes não estão meramente em mudança, mas sim a coadaptar-se à adaptação prévia do aprendiz a eles próprios. Então, a adaptação dos agentes ao ambiente pode mudá-lo a si mesmo duma forma que torna essa adaptação inválida. Isto é uma violação substancial das suposições básicas subjacentes às técnicas mais tradicionais de Aprendizagem Automática.

6.4. Aprendizagem na afinação de parâmetros

Como já referido no Capítulo 3, a afinação de parâmetros ainda é muitas vezes realizada sem a utilização de procedimentos automáticos e tem recebido pouca atenção por parte dos investigadores. Só recentemente começaram a surgir, na literatura, metodologias consolidadas para tratar este problema de forma eficiente e eficaz, embora poucas recaiam na aplicação de aprendizagem.

Além dos algoritmos de *Racing* (subsecção 6.4.1) e do Raciocínio baseado em Casos (subsecção 6.4.2), descritos com mais detalhe por terem sido utilizados neste trabalho para tratar o problema de afinação de parâmetros de Meta-heurísticas, é possível encontrar outras abordagens na literatura. Dobslaw (2010) apresentou uma *framework* para a simplificação e padronização de parâmetros de Meta-heurísticas através da aplicação de DOE e Redes Neuronais Artificiais. A fase de treino era dividida na fase de experiência e na fase de aprendizagem, fases estas que estavam ligadas entre si, e correspondiam à maior parte do gasto de tempo computacional. Smith-Miles (2008) apresentou e discutiu uma metodologia para resolver o problema de seleção de algoritmos do ponto de vista do problema de aprendizagem, integrando conceitos de meta-aprendizagem. Stoean *et al.* (2009) apresentaram uma metodologia combinada para resolver o problema de afinação de parâmetros em Meta-heurísticas e usam uma abordagem de amostragem para gerar um grande conjunto diverso de configurações para as variáveis a serem afinadas. Este conjunto é então sujeito a uma regressão através do uso de uma abordagem evolucionária de *Support Vector Machines*. Zennaki e Ech-Cherif (2010) usaram técnicas de Aprendizagem Automática, tais como Árvores de Decisão, aprendidas a partir de um grupo de várias

soluções de instâncias geradas aleatoriamente. Estas instâncias-soluções são usadas repetidamente para prever a qualidade de uma solução para uma dada instância do problema de otimização combinatória, a qual é usada para afinar e orientar a Meta-heurística para áreas mais promissoras do espaço de pesquisa. Lessmann *et al.* (2011) apresentaram uma abordagem baseada em *Data Mining* para lidar com o problema de afinação de um algoritmo de *Particle Swarm Optimization*. Os autores propuseram um sistema híbrido que emprega modelos de regressão para aprender valores adequados para os parâmetros com base em movimentos anteriores da Meta-heurística.

6.4.1. Algoritmos de *Racing*

A abordagem mais intuitiva para resolver o problema de afinação de parâmetros é através de força-bruta, que consiste em atribuir uma parte igual de poder computacional a cada configuração candidata, i.e., efetuar o mesmo número de experiências em cada uma delas. De modo a definir um algoritmo que implemente a abordagem de força-bruta, é necessário discutir um elemento importante: como estimar o desempenho esperado de uma configuração candidata usando um número finito de experiências, que foram afetadas para a sua avaliação? Em particular, a questão preocupa-se com o número de instâncias que devem ser consideradas para a avaliação e, como consequência, o número de execuções a ser efetuado em cada uma dessas instâncias.

No entanto, uma abordagem de força-bruta não é, claramente, a melhor solução para o problema de afinação de parâmetros (Birattari, 2009). Uma forma mais refinada e eficiente para afinação de Meta-heurísticas pode ser obtida através da racionalização da avaliação dos candidatos e através da libertação daqueles que aparentam ser menos promissores durante o processo de avaliação. Esta afirmação resume a abordagem de *Racing* aplicada ao problema de afinação de parâmetros em técnicas de otimização. A informação que se segue foi baseada em (Birattari, 2009; Hamadi *et al.*, 2012).

6.4.1.1. Perspetiva história e motivação

Na década de 1990, Maron e Moore (1993) propuseram o método *Hoeffding Race*, com o objetivo de acelerar a seleção de modelos em problemas de Aprendizagem Supervisionada. A ideia principal que serviu por base a este método foi apresentada na tese de Maron (1994) e pretende selecionar o conjunto de melhores parâmetros estruturais para um método baseado em memória. A medida de desempenho usada foi o erro quadrático

médio, calculado através do método de validação cruzada "*leave-one-out*", que consiste em prever cada exemplo do conjunto disponível com base em todos os outros exemplos. Por ser um procedimento muito dispendioso computacionalmente, em vez de se treinar *N* vezes a estrutura do modelo "*leave-one-out*", a ideia principal introduzida pelo *Hoeffding Race* permite acelerar a pesquisa pela melhor estrutura de modelos, descartando os candidatos inferiores assim que são obtidas evidências suficientes contra eles. Esta ideia caracteriza toda a classe de algoritmos de *Racing*.

De facto, a avaliação da medida "*leave-one-out*" relativa ao candidato genérico pode ser realizada incrementalmente. Por exemplo, considere-se a média dos *N* erros quadráticos, cada uma relativa aos *N* exemplos do conjunto de dados. Esta quantidade pode ser aproximada pela média calculada em qualquer subconjunto desses erros. É importante notar que, independentemente do tamanho da amostra considerada, a média calculada nesta amostra é uma estimativa imparcial do erro quadrático médio, e que, além disso, a variância de tal estimativa decresce com o tamanho. Assim que o cálculo procede e a estimativa da medida "*leave-one-out*" dos candidatos fica mais nítida, pode ser adotado um teste estatístico de hipóteses para permitir decidir se a diferença observada na estimativa "*leave-one-out*" dos candidatos é ou não significativa. Neste caso, os candidatos inferiores são descartados da corrida e não serão mais avaliados. A Figura 6.1 retrata isto mesmo, à medida que o número de exemplos (ou instâncias) aumenta, o número de candidatos vai sendo reduzido.

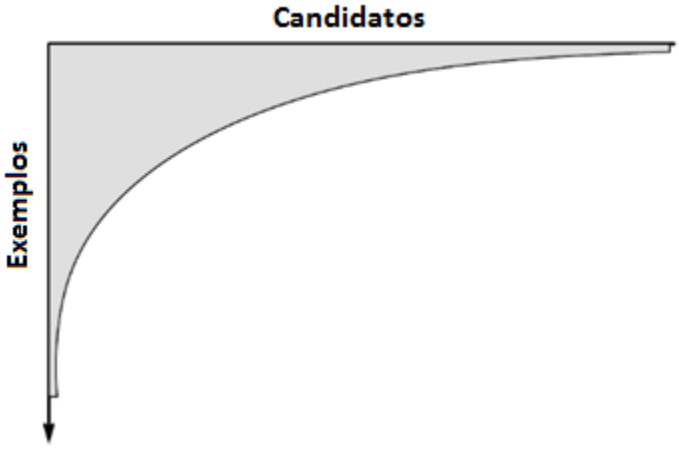

Figura 6.1 – Representação gráfica da computação envolvida no *Hoeffding Race* (baseado em (Birattari, 2009)

O método de *Racing* original, o *Hoeffding Race*, adotou um teste estatístico baseado na fórmula de Hoeffding (1963) relativa à confiança na média empírica de *k* números positivos, cuja amostra foi realizada de forma independente a partir da mesma distribuição, quando se conhece um limite superior.

O *Hoeffding Race* começa com um conjunto de candidatos e procede considerando iterativamente os exemplos de um dado conjunto de dados de tamanho *N*. Num dado passo *k* do processo iterativo, o exemplo *k* do conjunto de dados é estendido e todos os candidatos da corrida são chamados para prever o *output* com base em todos os outros exemplos. Para um candidato genérico, o erro é usado para atualizar o erro quadrático médio estimado do método "*leave-one-out*". Depois de completar esse passo *k* e antes de passar ao próximo passo, todos os candidatos que tenham um limite inferior abaixo do limite superior do melhor candidato encontrado até ao momento são eliminados da corrida e deixam de ser considerados em futuras avaliações. À medida que a avaliação dos candidatos sobreviventes procede e mais exemplos são considerados no procedimento "*leave-one-out*", os intervalos em redor das estimativas empíricas ficam mais apertados e o procedimento de seleção fica mais refinado.

A ideia subjacente às abordagens de *Racing* é muito apelativa e, em (Moore e Lee, 1994), foram propostos alguns algoritmos baseados em diferentes testes estatísticos. Entre eles, o BRACE é baseado em estatística Bayesiana e implementa uma técnica estatística conhecida por *blocking* (Box *et al.*, 1978; Dean e Voss, 1999; Montgomery, 2008). Um plano de blocos é uma definição experimental que é possivelmente adotada quando dois ou mais candidatos têm de ser comparados e que melhoram a precisão com a qual a comparação é realizada. Um bloco é um conjunto de condições experimentais relativamente homogéneas sob as quais todos os candidatos são testados. No contexto da seleção do melhor candidato com base na validação "*leave-one-out*", a adoção de planos de blocos é particularmente natural e simples: cada um dos candidatos sobreviventes é testado nos mesmos exemplos, e cada exemplo é portanto um bloco no plano considerado.

No BRACE, durante a corrida, o algoritmo estima incrementalmente o valor médio e a variância de cada par de candidatos. Com base nestas estimativas, é efetuado um conjunto de testes estatísticos em cada passo do algoritmo para verificar se algum candidato é significativamente pior do que algum outro. Neste caso, o candidato inferior é descartado da corrida. Sob algumas condições gerais, o teste estatístico adotado no BRACE é equivalente ao *paired t-test* (Sheskin, 2003) realizado entre cada par de candidatos sobreviventes. Contrariamente ao teste adotado no *Hoeffding Race*, o *t-test* é um

procedimento paramétrico e baseia-se portanto em algumas assunções relativas às variáveis estocásticas envolvidas. O BRACE provou ser muito eficaz e é reportado como capaz de atingir melhores resultados do que o *Hoeffding Race* (Maron e Moore, 1997; Moore e Lee, 1994).

É de notar, no entanto, que o *Hoeffding Race* adota um teste não-paramétrico, mesmo se o teste adotado precisa de conhecimento de um limite no erro observado, e este aspeto reduz significativamente a variedade de aplicações do método. Por outro lado, o BRACE provou que a adoção de um plano de blocos é particularmente eficaz na definição da corrida.

6.4.1.2. O método *F-Race*

Se o tempo global disponível para a afinação dos parâmetros for T e cada experiência for executada por um tempo t, então o número total de experiências que podem ser realizadas corresponde a $M = T/t$. Numa abordagem de força-bruta, o poder computacional é atribuído uniformemente às diferentes θ configurações candidatas. Assim, cada uma delas é testada $N = M/\theta$ vezes de modo a estimar o desempenho esperado. A alocação ótima das N experiências permitidas a serem efetuadas em cada configuração consiste em executar cada configuração uma vez em N instâncias diferentes, de acordo com uma determinada distribuição.

Com base nestes elementos, é possível definir um algoritmo que implementa a abordagem de força-bruta. Neste algoritmo, N instâncias são selecionadas aleatoriamente de acordo com uma distribuição desconhecida. Todas as configurações candidatas para a Meta-heurística são testadas uma vez nas instâncias selecionadas. Com base nos resultados observados, para cada candidato determina-se uma estimativa do desempenho esperado. É então selecionado o candidato para o qual o desempenho esperado estimado seja o melhor. No Algoritmo 6.1 é apresentada uma descrição do algoritmo de força-bruta em pseudocódigo.

É importante notar que os requisitos de memória do algoritmo de força-bruta são particularmente limitados. Já que a única peça de informação que é necessário reter sobre cada configuração candidata é apenas uma estimativa do seu desempenho esperado (i.e., a média empírica dos resultados observados em N instâncias diferentes) e uma vez que a média dos N valores pode ser calculada incrementalmente, é apenas necessário guardar a

estimativa atual para cada candidato. Um *array* com o tamanho correspondente ao número de configurações candidatas guarda toda a informação do cálculo em processamento.

Algoritmo 6.1 – Algoritmo da abordagem de força-bruta (baseado em (Birattari, 2009)

```
     Entrada: M, Θ  // Número total de experiências e conjunto de configurações candidatas
     Saída: Melhor // Melhor configuração candidata
1    N ← M / tamanho(Θ) ; // Número de experiências para cada candidato
2    A ← array(tamanho(Θ)); // Array para guardar o desempenho esperado dos candidatos
3    NumIteracoes ← 1;
4    Repete
5           instancia ← gerarInstancia(); // Gerar uma instância
6           Para cada conf  ∈ Θ Faz // Percorrer todos os candidatos
7                  // Executar o candidato na instância gerada
8                  s ← correrExperiencia(conf, instancia);
9                  c ← avaliarSolucao(s); // Avaliar a solução obtida
10                 // Atualizar a estimativa do desempenho esperado para o candidato
11                 A[conf] ← atualizarDesempenho(A[conf], c, NumIteracoes);
12          Fim
13   Enquanto NumIteracoes++ ≤ N; // Repete o ciclo enquanto não atinge o critério de paragem
14   Melhor ← minimo(A);
15   Devolver Melhor;
```

A ideia principal subjacente a uma abordagem *Racing* é a avaliação do desempenho de uma configuração candidata poder ser realizada incrementalmente. Todas as condições são satisfeitas por definição, considerando simplesmente as instâncias sequenciais que aparecem no fluxo de instâncias, e em cada uma executar uma vez a configuração da Meta-heurística. Além disso, seguindo tal procedimento que executa uma única corrida em cada instância, garante-se que a estimativa obtida tem a menor variância possível.

Dada a possibilidade de construir a sequência de estimativas para cada configuração candidata, um algoritmo de *Racing* pode ser definido como aquele que constrói incrementalmente, e em paralelo, tais sequências para todas as configurações candidatas e, logo que sejam obtidas evidências suficientes que uma determinada configuração seja pior do que as restantes, este é descartada de avaliações futuras. Um algoritmo de *Racing* gera então uma sequência de conjuntos aninhados de configurações candidatas, sendo que o conjunto inicial é igual à totalidade de configurações candidatas. Em cada passo iterativo é possível descartar algumas configurações que parecem ser sub-ótimas com base na informação disponível.

Num dado passo iterativo existe um conjunto de candidatos e é considerada uma nova instância. Cada candidato é testado nessa instância e cada custo observado é adicionado ao respetivo *array* para formar o conjunto de diferentes *arrays*, cada um

pertencente a uma configuração candidata. Esse passo iterativo termina definindo um novo conjunto no qual foi retirada a configuração ou configurações que parecem ser sub-ótimas com base em algum teste estatístico que compara todos os *arrays* entre si. O procedimento descrito é iterado e termina quando todas as configurações exceto uma forem descartadas, ou quando um número máximo de instâncias é atingido, ou mesmo quando foram efetuadas um número máximo de *M* experiências. O pseudocódigo do algoritmo de *Racing* genérico é apresentado no Algoritmo 6.2.

Algoritmo 6.2 – Algoritmo de *Racing* (baseado em (Birattari, 2009)

```
    // Número total de experiências, conjunto de configurações candidatas e n° máximo de instâncias
    Entrada: M, Θ, MaxInst
    Saída: Melhor // Melhor configuração candidata
1   Exp ← 0; // Número de experiências até ao momento
2   Inst ← 0; // Número de instâncias até ao momento
3   A ← array(tamanho(Θ), MaxInst); // Array para guardar o desempenho dos candidatos
4   α ← Θ; // Candidatos sobreviventes
5   Repete
6           instancia ← gerarInstancia(); // Gerar uma instância
7           Para cada conf  ∈ α Faz // Percorrer todos os candidatos sobreviventes
8                   // Executar o candidato na instância gerada
9                   s ← correrExperiencia(conf, instancia);
10                  Exp++; // Incrementar número de experiências
11                  // Atualizar a estimativa do desempenho esperado para o candidato
12                  A[Inst, conf] ← atualizarDesempenho(s);
13          Fim
14          // Eliminar os candidatos inferiores de acordo com o teste estatístico
15          α ← eliminarCandidatos(α, A);
16  // Repete o ciclo enquanto o critério de paragem não é atingido
17  Enquanto (Exp + tamanho(α) ≤ M) e (Inst++ < MaxInst);
18  Melhor ← minimo(A, α);
19  Devolver Melhor;
```

A aparente vantagem das abordagens de *Racing* sobre a força-bruta baseia-se no facto de fornecer uma melhor afetação dos recursos computacionais entre as configurações candidatas: em vez de desperdiçar tempo computacional para estimar de forma precisa o desempenho dos candidatos inferiores, a abordagem de *Racing* foca-se nos mais promissores e obtém uma estimativa com menor variância para estes. Tal aspeto permite uma seleção mais informada do melhor candidato. A Figura 6.2 propõe uma representação gráfica das diferentes estratégias adotadas pelas duas abordagens para organizar a avaliação das configurações candidatas. Nas abordagens de *Racing*, um candidato é descartado de futuras avaliações assim que existem evidências suficientes de que não é o melhor. Ao longo do processo de avaliação, a abordagem de *Racing* foca-se cada vez mais

nos candidatos mais promissores. Por outro lado, a abordagem de força-bruta testa todos os candidatos no mesmo número de instâncias.

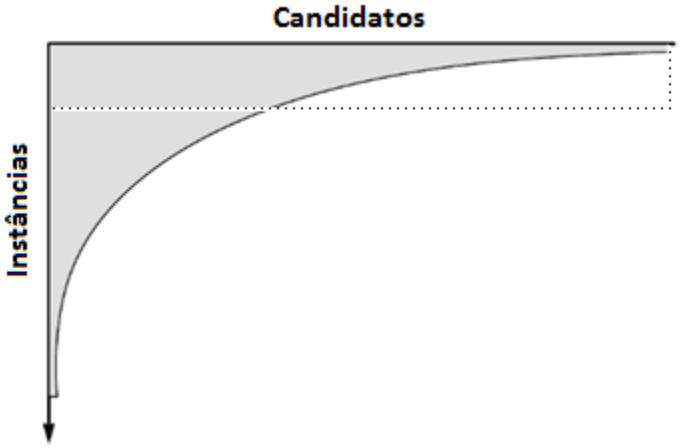

Figura 6.2 – Representação gráfica das diferentes estratégias adotadas pelas abordagens de *Racing* e pela abordagem de força-bruta (a tracejado) para afetação de poder computacional na avaliação de candidatos (baseado em (Birattari, 2009))

Durante a corrida, é necessário manter algumas estatísticas dos resultados obtidos pelos candidatos sobreviventes (Mendenhall *et al.*, 1981). A quantidade de informação a ser guardada depende do teste estatístico específico a ser usado para decidir se as diferenças observadas no desempenho dos candidatos são significativas ou não. Os requisitos de memória para uma abordagem de *Racing* são então delimitados por M vezes a quantidade de memória necessária para armazenar o custo da solução encontrada numa única corrida da Meta-heurística (tipicamente muito menos, uma vez que os resultados obtidos pelas configurações descartadas antes da eliminação podem ser descartados também).

O método *F-Race* (Birattari *et al.*, 2002) é uma abordagem de *Racing* que se inspira e integra num único algoritmo as melhores características do *Hoeffding Race* e do BRACE. Tal como já referido, o *Hoeffding Race* adota uma abordagem não-paramétrica mas não considera o *blocking*. Por outro lado, o BRACE adota o *blocking* mas descarta a definição não-paramétrica em detrimento de uma abordagem Bayesiana. O *F-Race* é baseado no teste de Friedman (*Friedman two-way analysis of variance by ranks*), que implementa um plano de blocos de uma forma extremamente natural e é, ao mesmo tempo, não-paramétrico (Conover, 1999).

O teste de Friedman considera a estatística de Conover (1999). Para dar uma descrição deste teste, assuma-se que o *F-Race* atingiu o passo k e que ainda estão na

corrida *k-1* configurações. O teste de Friedman assume que os custos observados até ao momento para as configurações ainda na corrida são a realização de *k* mutuamente independentes variáveis aleatórias, designadas de blocos. Cada bloco corresponde aos resultados computacionais obtidos numa instância por cada configuração ainda na corrida no passo *k*.

Birattari (2009) descreveu que, como complemento ao *F-Race,* pode ser utilizado o teste de Wilcoxon (*Wilcoxon matched-pairs signed-ranks*) (Conover, 1999) quando existem apenas dois candidatos na corrida, uma vez que este se revelou mais robusto e eficiente.

No *F-Race*, o *ranking* desempenha um importante papel duplo. Primeiro, está ligado com a natureza não-paramétrica do teste baseado em *ranking*. O principal mérito da análise não-paramétrica é que não requer a formulação de hipóteses na distribuição das observações. É precisamente na adoção de um teste estatístico baseado em *ranking* que diverge de trabalhos anteriormente publicados.

Birattari (2009) mencionou alguns factos amplamente aceites acerca de testes de hipóteses paramétricos e não-paramétricos. O autor referiu que, quando as hipóteses formuladas forem satisfeitas, os testes paramétricos têm um poder superior do que os não-paramétricos e requerem normalmente muito menos tempo de computação. Igualmente, quando está disponível uma grande quantidade de dados, as hipóteses para a aplicação de testes paramétricos tendem a ser cumpridas em virtude do teorema do limite central.

6.4.1.3. Aplicações de métodos de *Racing*

Durante o desenvolvimento deste trabalho de doutoramento, não foi possível identificar muitas aplicações de abordagens de *Racing* na resolução de problemas de escalonamento de produção. No entanto, foi possível encontrar aplicações diversas destas abordagens noutros domínios de aplicação. Besten (2004) usou uma abordagem de *Racing* para afinar um algoritmo de Pesquisa Local Iterativa na resolução de problemas de escalonamento de produção determinísticos com penalização de atrasos. Yuan e Gallagher (2004) discutiram o uso dos algoritmos de *Racing* na avaliação empírica de algoritmos evolucionários e introduziram o *A-Race*, uma instância paramétrica de *Racing* baseada no método de análise de variância. Chiarandini (2005), no seu trabalho de Doutoramento, adotou o método de *Racing* para afinar os parâmetros de algoritmos para problemas de otimização combinatória com muitas restrições. Em (Birattari *et al.*, 2007), o *F-Race* foi adotado como um módulo de um algoritmo de Colónia de Formigas para tratar problemas de

otimização combinatória sob incertezas. Socha (2008) usou o *F-Race* para afinar os parâmetros de um algoritmo de Colónia de Formigas para problemas de otimização contínua e de variáveis mistas e Nouyan (2008) usou-o para afinar os parâmetros de um sistema de *Swarm Robotics.* O *F-Race* foi também usado por Balaprakash *et al.* (2009) para afinar os parâmetros de um algoritmo de Pesquisa Local para otimização combinatória sob incertezas, na resolução do problema do Caixeiro-Viajante.

Na indústria, as abordagens de *Racing* foram consideradas por Becker (2004) num estudo de viabilidade, que tinha o objetivo de otimizar um programa computacional industrial para a resolução de problemas de otimização de rotas de veículos e problemas de escalonamento, desenvolvidos pela companhia SAP.

As abordagens de *Racing* têm também sido usadas em conjunto com outros métodos. Yuan e Gallagher (2007) descreveram como o *Racing* pode ser combinado com o Meta-EA de modo a reduzir o número de combinações de parâmetros e o número de testes necessários. Smit e Eiben (2009) usaram o *Racing* como um módulo de um método cujo objetivo é estimar a relevância e calibrar os valores dos parâmetros de um algoritmo evolucionário.

6.4.2. Raciocínio baseado em Casos

O Raciocínio baseado em Casos é uma metodologia de Inteligência Artificial com o objetivo de resolver novos problemas através do uso da informação acerca das soluções de problemas anteriores similares e tem sido alvo de uma grande atenção por parte da comunidade científica, ao longo das últimas duas décadas (Aamodt e Plaza, 1994; Kolodner, 1993).

Como já referido anteriormente, enquadra-se na Aprendizagem baseada em Instâncias e opera sob a premissa de que os problemas similares podem ter soluções similares (Beddoe *et al.*, 2009). Leake (1996) referiu que *"no Raciocínio baseado em Casos, são geradas novas soluções não por encadeamento, mas pela recuperação dos casos mais relevantes da memória e pela sua adaptação para novas soluções"*.

Os problemas resolvidos anteriormente no sistema e as suas soluções (ou estratégias associadas) são memorizados como casos e guardados num repositório, a base de casos, para poderem ser reutilizados no futuro (Beddoe *et al.*, 2009; Burke *et al.*, 2003). Em vez de se definir um conjunto de regras ou linhas gerais, um sistema de Raciocínio baseado em Casos resolve o novo problema através da reutilização de problemas similares

anteriormente resolvidos (Petrovic *et al.*, 2007). Normalmente, é necessário adaptar as soluções para que um novo problema as use, e os novos problemas resolvidos devem ser guardados e a base de casos atualizada (Burke *et al.*, 2003).

A informação descrita nas duas próximas subsecções foi baseada na informação de Kolodner (1993) e Aamodt e Plaza (1994).

6.4.2.1. Perspetiva histórica e motivação

A origem do Raciocínio baseado em Casos pode ser encontrada nos trabalhos de Roger Schank em memória dinâmica (Schank, 1982), mas também no estudo do raciocínio analógico (Gentner, 1983).

O primeiro sistema que usou a técnica de Raciocínio baseado em Casos foi o CYRUS, desenvolvido por Kolodner (1993), que se baseava no modelo de memória dinâmica de Schank (1982). Consistia num sistema de pergunta-resposta com conhecimento das várias viagens e encontros de Cyrus Vance, ex-Secretário de Estado dos Estados Unidos da América. O modelo de memória de casos desenvolvido serviu mais tarde como base para outros sistemas de Raciocínio baseado em Casos.

Outra base para o Raciocínio baseado em Casos surgiu com o grupo de Bruce Porter (Porter e Bareiss, 1986), que inicialmente abordou o problema de Aprendizagem Automática para tarefas de classificação. Este grupo desenvolveu o sistema PROTOS que enfatizou a integração de conhecimento geral de domínio e conhecimento específico dos casos numa estrutura de representação unificada.

O Raciocínio baseado em Casos é um paradigma de resolução de problemas bastante diferente, em muitos aspetos, de outras abordagens de Inteligência Artificial. Em vez de se basear somente em conhecimento geral de um domínio do problema, o Raciocínio baseado em Casos é capaz de utilizar conhecimento específico de situações concretas de um problema, previamente conhecidas (casos). Um novo problema é resolvido através da procura de um caso passado similar, e da sua reutilização na nova situação. Outra importante diferença baseia-se no facto do Raciocínio baseado em Casos ser uma abordagem para aprendizagem incremental e sustentada, uma vez que uma nova experiência é retida cada vez que um problema é resolvido, ficando imediatamente disponível para problemas futuros.

O Raciocínio baseado em Casos é então, basicamente, a resolução de um novo problema através da recordação de situações similares anteriores e através da reutilização da informação e conhecimento dessa situação. Considere-se, por exemplo, o caso dum médico a examinar um paciente no seu consultório. Considerando os sintomas que o paciente apresenta, o médico relembra-se de pacientes anteriores com os mesmos sintomas e usa o mesmo procedimento para realizar o diagnóstico e aplica o mesmo tratamento neste paciente.

Tal como o exemplo anterior demonstra, a reutilização de casos passados é um método poderoso e frequentemente aplicado pelos humanos para a resolução de problemas. Parte do fundamento para a abordagem baseada em casos é a sua plausibilidade psicológica. Ross (1989) apresentou algumas evidências empíricas para o papel dominante de situações específicas experienciadas anteriormente (casos) na resolução de problemas dos humanos. Schank (1982) desenvolveu uma teoria de aprendizagem e memória baseada na retenção de experiência em estruturas de memória evolutiva dinâmica (base de casos). Gentner (1983) estudou a resolução de problemas por analogia que também mostraram o uso de experiência passada na resolução de novos e diferentes problemas.

Na terminologia de Raciocínio baseado em Casos, um **caso** denota normalmente uma situação do problema. Uma situação previamente experienciada, que foi capturada e aprendida para que possa ser reutilizada na resolução de problemas futuros, é referida como um **caso passado**. Um **novo caso**, ou caso não resolvido, é a descrição de um novo problema para ser resolvido. Assim, o Raciocínio baseado em Casos é um processo cíclico e integrado de resolução de problemas, que aprende com a sua experiência na resolução de novos casos.

Kolodner (1993) referiu que "*um caso é um pedaço contextualizado de conhecimento representando uma experiência que ensina uma lição fundamental para a consecução dos objetivos do sistema*". Segundo Burke *et al.* (2003), um caso consiste em duas partes principais: o problema em si, representado de uma certa forma para descrever as condições sobre as quais deve ser recuperado, e a solução que foi usada para lidar com o problema. Pode ainda conter o resultado, ou seja, sucesso, falha, ou uma descrição do estado, e algumas vezes o contexto em que o caso foi gerado, ou o contexto em que o caso possa ser usado no futuro (Beddoe *et al.*, 2009).

6.4.2.2. O ciclo dos 4 “REs”

O paradigma de Raciocínio baseado em Casos cobre uma variedade de métodos diferentes para organizar, recuperar, utilizar e indexar o conhecimento retido em casos anteriores. Estes métodos incluem a identificação da situação do problema, a pesquisa de casos anteriores similares ao novo caso, a utilização do caso mais similar para sugerir uma solução para o novo problema, a avaliação da solução proposta, e a atualização do sistema através da aprendizagem da experiência.

Os casos podem ser mantidos como experiências concretas, ou um conjunto de casos similares devem dar origem a um caso generalizado. Podem ser armazenados como unidades de conhecimento separadas, ou divididas em subunidades e distribuídas dentro da estrutura de conhecimento. Os casos podem ainda ser indexados dentro de uma estrutura de índices plana ou hierárquica.

A solução de um caso anterior pode ser aplicada diretamente ao problema atual, ou modificada de acordo com as diferenças entre os dois casos. A seleção de casos, adaptação de soluções, e aprendizagem a partir de uma experiência pode ser guiada e suportada por um modelo de conhecimento geral de domínio, através de conhecimento mais compilado e raso, ou ser baseada apenas numa similaridade aparente e sintática.

Os métodos de Raciocínio baseado em Casos podem ser puramente autossuficientes e automáticos, ou podem interagir com o utilizador para apoio e orientação das suas ações. Alguns métodos assumem uma quantidade significativa de casos amplamente distribuídos na sua base de casos, enquanto outros se baseiam em conjuntos mais limitados de casos típicos. Os casos anteriores podem ser recuperados e avaliados sequencialmente ou em paralelo.

Assim, um ciclo geral de Raciocínio baseado em Casos pode ser descrito através dos quatro processos seguintes, conhecidos como “*os quatro REs*” (Aamodt e Plaza, 1994; Beddoe e Petrovic, 2006; Beddoe *et al.*, 2009):

- **Recuperar** o caso (ou casos) mais similar(es). A recuperação de casos implica: identificar um problema, o que pode envolver a simples observação das suas características mas também abordagens mais elaboradas de modo a tentar perceber o problema dentro do seu contexto; retornar um conjunto de casos suficientemente similares com o novo caso, de acordo com uma similaridade mínima; e selecionar o melhor caso deste conjunto de casos.

- **Reutilizar** a informação e o conhecimento do caso recuperado para resolver o problema. A reutilização da solução do caso recuperado foca-se em dois aspetos: as diferenças entre o caso passado e o caso atual; e na identificação de partes do caso recuperado que possam ser transferidas para o novo caso, o que pode ser realizado através de uma cópia direta ou através de adaptação da solução para o novo caso.
- **Rever** a solução proposta. A fase de revisão consiste em aplicar a solução sugerida pela fase de reutilização num ambiente de execução, avaliar o resultado obtido, e aprender a partir do sucesso (passagem à próxima fase) ou efetuar uma reparação da solução através da utilização de conhecimento específico do domínio do problema.
- **Reter** as partes da experiência que se revelarem úteis para resoluções futuras de problemas. A aprendizagem a partir do sucesso ou falha da solução proposta é desencadeada através do resultado da avaliação e possível reparação. A fase de retenção envolve selecionar qual a informação que é importante guardar, de que forma retê-la, como indexar o caso para uma recuperação futura, e como integrar o novo caso na estrutura de memória.

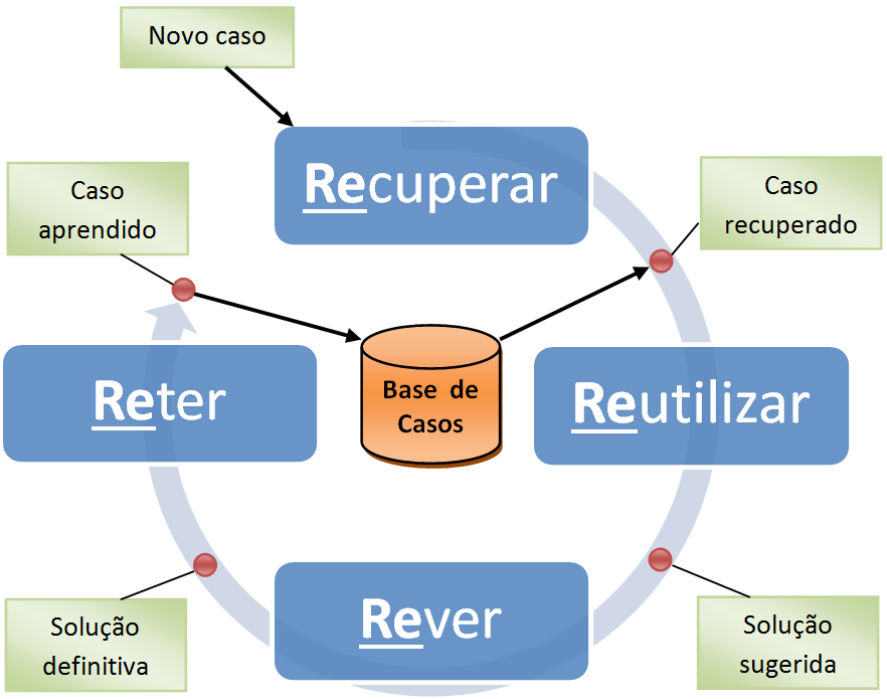

Figura 6.3 – O ciclo do Raciocínio baseado em Casos (adaptada de (Aamodt e Plaza, 1994))

A Figura 6.3 ilustra o ciclo de funcionamento do Raciocínio baseado em Casos, onde uma descrição inicial de um problema define um novo caso, que é usado para recuperar um caso da base de casos. O caso recuperado é combinado com o novo caso, através da reutilização, para dar origem a uma solução sugerida para o problema inicial. Através do processo de revisão, a solução é testada, por exemplo, através da aplicação num ambiente de execução, e possivelmente reparada se falhar. Durante o processo de retenção, a experiência útil é retida para reutilização futura, e a base de casos é atualizada com o novo caso aprendido, ou através da modificação de alguns casos existentes.

A tarefa de **recuperação** de casos começa com uma descrição parcial do problema, e termina quando é encontrado o caso mais similar na base de casos. As suas subtarefas são referidas como **Identificação**, **Correspondência**, e **Seleção**, executadas nesta ordem. A Identificação surge basicamente com um conjunto de características relevantes para o problema. O objetivo da Correspondência é retornar um conjunto de casos que são suficientemente similares com o novo caso, dada uma similaridade mínima. A Seleção, tal como nome indica, pretende selecionar o melhor caso do conjunto existente.

Identificar um problema pode envolver a simples observação das suas características de entrada, mas muitas vezes, e particularmente para métodos de conhecimento intensivo, aplicam-se abordagens mais elaboradas, nas quais são realizadas tentativas para "perceber" o problema no seu contexto. Perceber um problema envolve a filtragem das características ruidosas do problema, de modo a inferir outras relevantes ao problema, para verificar se os valores das características fazem sentido no contexto, para gerar expectativas para outras características, etc. Estas outras características podem ser inferidas através do uso de um modelo de conhecimento geral, ou através da recuperação de descrições similares ao problema, a partir da base de casos, e usar características desse caso como características esperadas.

A tarefa de encontrar uma boa correspondência é tipicamente dividida em duas subtarefas: um processo inicial de correspondência que recupera um conjunto plausível de casos candidatos e um processo mais elaborado de seleção do melhor caso do conjunto. O processo de encontrar um conjunto de casos é realizado através do uso das características do problema como índices para a base de casos de uma forma direta ou indireta. A literatura refere três formas de recuperação de casos (Aamodt e Plaza, 1994): através do uso direto dos apontadores dos índices das características do problema, através da pesquisa numa estrutura de índices, ou através da pesquisa num modelo de conhecimento geral de domínio. Os casos podem ser recuperados apenas a partir das características de entrada,

ou também a partir das características inferidas das entradas. Os casos que correspondem a todas as características de entrada são bons candidatos para a correspondência, mas, dependendo da estratégia seguida, os casos que correspondem a uma certa porção das características do problema (de entrada ou inferidas) também podem ser recuperados. É necessária uma forma de aceder ao nível de similaridade. Uma medida de similaridade é definida normalmente por uma fórmula para calcular a similaridade entre casos prévios e o novo caso (Burke *et al.*, 2003). Os casos anteriores mais similares são recuperados para o novo caso. O desenvolvimento desta medida de similaridade para problemas complexos do mundo real tem sido identificado como um dos maiores desafios de investigação nesta área.

Na Seleção, é selecionado o melhor caso a partir do conjunto de casos similares. Este processo pode ser efetuado durante o processo de correspondência inicial, mas muitas vezes é retornado um conjunto de casos, ao invés de um único caso. O caso mais similar é normalmente determinado através da avaliação mais profunda do nível de correspondência inicial. O processo de seleção gera tipicamente consequências e expectativas para cada caso recuperado, e tenta avaliar as consequências e justificar as expectativas. Isto pode ser realizado através da utilização do modelo de conhecimento geral de domínio do sistema, ou questionando o utilizador para confirmar ou para introduzir informação adicional. Os casos são eventualmente classificados recorrendo a alguns critérios métricos ou de classificação.

A **reutilização** da solução do caso recuperado foca-se em dois aspetos: as diferenças entre o caso passado e o caso atual, e na identificação de partes do caso recuperado que possam ser transferidas para o novo caso. Em tarefas de classificação simples, as diferenças são ignoradas (as diferenças são consideradas não relevantes enquanto as similaridades são consideradas relevantes) e a solução do caso recuperado é transferida para o novo caso como a sua solução. Este é um tipo trivial de reutilização. No entanto, outras abordagens têm em consideração as diferenças entre os dois casos, sendo que a parte reutilizada não pode ser diretamente transferida para o novo caso, pois requer um processo de adaptação que tem em conta essas diferenças.

A literatura refere duas formas de reutilizar casos passados:

- Reutilizar a solução do caso passado, onde a solução passada não é diretamente a solução para o novo caso, mas existe algum conhecimento que permite transformar a antiga solução na solução para o novo caso.
- Reutilizar o método passado que constrói a solução, onde é observada a forma como o problema foi resolvido no caso recuperado.

O caso recuperado detém informação sobre o método usado para a resolução do problema incluindo uma justificação sobre os operadores usados, sub-objetivos considerados, alternativas geradas, falhas, etc. O método recuperado é então reutilizado para o novo problema no novo contexto.

Quando uma solução, gerada pela fase de Reutilização, não é correta, surge uma oportunidade de aprender através da falha. Esta fase é chamada de fase de **revisão** e consiste em avaliar a solução do caso gerada pela reutilização. Se esta solução for bem-sucedida, é realizada a aprendizagem a partir do sucesso (passagem à próxima fase), senão é efetuada uma reparação da solução através da utilização de conhecimento específico do domínio do problema.

A avaliação aplica a solução proposta num ambiente de execução e avalia o resultado. Este é normalmente um passo fora do sistema de Raciocínio baseado em Casos, uma vez que requer a execução do problema numa aplicação. Os resultados desta execução podem demorar algum tempo a surgir, dependendo do tipo de aplicação. A solução pode ainda ser aplicada num programa de simulação que é capaz de gerar a solução correta.

A reparação de casos envolve a deteção dos erros da solução atual e a recuperação ou geração das explicações para esses erros. As explicações são usadas de modo a ser possível modificar a solução dum modo em que as falhas detetadas anteriormente não ocorram. A solução revista pode ser retida diretamente, ou pode ser avaliada e reparada novamente.

Finalmente, a fase de **retenção** permite incorporar no conhecimento existente o que é útil reter da resolução do novo problema. A aprendizagem a partir do sucesso ou falha da solução proposta é desencadeada através do resultado da avaliação e possível reparação. Esta fase envolve selecionar a informação a reter do caso, de que forma deve ser retida, como indexar o caso para uma recuperação futura em casos similares, e como integrar o novo caso na estrutura de memória.

No Raciocínio baseado em Casos, a base de casos é sempre atualizada, independentemente da forma como o problema foi resolvido. Se foi resolvido através do uso de um caso anterior, o novo caso pode ser construído com base neste. Se o problema foi resolvido por outros métodos, incluindo a requisição de informação ao utilizador, é necessário construir um novo caso completo para ser armazenado. Em qualquer das

situações, é necessário tomar uma decisão sobre o que usar como fonte de aprendizagem. Podem também ser incluídas no novo caso justificações acerca das soluções seguidas.

O problema de indexação é um problema central, bastante focado no Raciocínio baseado em Casos, e consiste em decidir quais os tipos de índices a usar em futuras recuperações de casos, bem como definir a estrutura do espaço de pesquisa dos índices. Os índices diretos evitam o segundo passo, mas requerem de qualquer forma o problema de identificar o tipo de índices a usar. Uma solução trivial para este problema é o uso de todas as características de entrada como índices.

O último passo da atualização da base de conhecimento é realizado através da integração do novo caso. Se não foi construído nenhum novo caso, nem nenhum conjunto de índices de raiz, a tarefa de integração corresponde ao passo principal do processo de retenção. Através da modificação dos índices dos casos existentes, os sistemas de Raciocínio baseado em Casos podem aprender para se tornarem melhores avaliadores de similaridade de casos. A afinação dos índices existentes é uma parte importante da aprendizagem, uma vez que a importância de um índice para um caso particular é ajustada devido ao sucesso ou falha da utilização de um caso passado para a resolução do problema. Para as características que se revelaram relevantes no âmbito da recuperação de um caso bem-sucedido, a associação com o caso é fortalecida, enquanto é enfraquecida para características que levaram a um caso mal sucedido.

Um sistema de Raciocínio baseado em Casos é bastante dependente da estrutura e conteúdo da sua base de casos. Desde que um problema é resolvido, através da utilização da experiência passada adequada para a resolução do novo problema, o processo de recuperação precisa de ser eficaz e razoavelmente eficiente. Adicionalmente, uma vez que a experiência de um problema solucionado tem de ser retida na base de casos, essa integração de um novo caso em memória deve ser igualmente eficaz e eficiente. No Raciocínio baseado em Casos existe o problema da representação de casos que consiste principalmente em decidir como a base de casos deve ser organizada e indexada para uma recuperação e reutilização eficaz. Uma dificuldade adicional refere-se à forma de integrar a estrutura da base de casos num modelo de conhecimento geral de domínio, na medida em que tal conhecimento é incorporado.

6.4.2.3. Aplicações em Escalonamento

No domínio do problema de Escalonamento, alguns investigadores têm recorrido ao Raciocínio baseado em Casos para usar conhecimento/experiência passada na resolução de novos problemas. Burke *et al.* (2003) referiram que o Raciocínio baseado em Casos é uma abordagem adequada para Sistemas de Escalonamento periciais e enfatizaram um potencial de investigação na área do Escalonamento Dinâmico.

Na generalidade, as aplicações de Raciocínio baseado em Casos no domínio do problema de Escalonamento podem ser classificadas em três categorias (Petrovic *et al.*, 2007): reutilização de algoritmos, reutilização de operadores e reutilização de soluções.

A afirmação de base subjacente aos sistemas de Raciocínio baseado em Casos para Escalonamento que fazem **reutilização de algoritmos** baseia-se no pressuposto de que uma abordagem que se revelou eficaz na resolução de um problema específico terá bastante probabilidade de ser eficaz na resolução de um problema similar. Neste tipo de sistemas, um caso consiste na representação do problema e num algoritmo conhecido que seja eficaz para a sua resolução. Schmidt (1998) especificou uma estrutura de Raciocínio baseado em Casos para escolher o método mais apropriado para a resolução de um dado problema de Escalonamento de tarefas em máquinas de produção. Um caso guardava problemas de Escalonamento nas máquinas, enquanto os casos na base de casos eram organizados num grafo onde se visualizavam as relações entre os casos. Um vértice num grafo representava um grupo de problemas de Escalonamento para o qual tinha sido sugerido um dado algoritmo. Schirmer (2000) implementou um sistema de Raciocínio baseado em Casos para a seleção de algoritmos de Escalonamento na resolução de problemas de escalonamento de projetos. O autor mostrou experimentalmente que alguns algoritmos de Escalonamento funcionam melhor do que outros algoritmos em algumas instâncias de problemas, uma vez que o sistema proposto pode selecionar com sucesso um algoritmo que supera um determinado número de Meta-heurísticas. No entanto, Petrovic *et al.* (2007) referiram que as partes constituintes destas duas abordagens, ou seja a representação do caso (através de pares atributo-valor) e os casos recuperados, têm de ser implementadas especialmente para o domínio do problema a resolver, o que faz com que não possam ser facilmente adaptadas e aplicadas a outros problemas.

Nos sistemas de Raciocínio baseado em Casos para Escalonamento que fazem **reutilização de operadores**, um caso descreve um contexto no qual um operador de escalonamento, que previamente provou ser útil, é usado para reparar/adaptar um plano de

escalonamento de forma a melhorar a sua qualidade, em termos de satisfação de restrições (Beddoe *et al.*, 2009). Beddoe e Petrovic (2006) apresentaram uma metodologia de reutilização de operadores para resolver problemas de escalonamento, onde um Algoritmo Genético foi combinado com Raciocínio baseado em Casos de modo a selecionar e ponderar recursos. Burke *et al.* (2006b) utilizaram este método para resolver problemas de escalonamento de horários, onde as situações bem-sucedidas são gravadas numa base de casos juntamente com a informação heurística e os casos que não provaram ser úteis na resolução foram descartados. Beddoe *et al.* (2009) desenvolveram um sistema de Raciocínio baseado em Casos para problemas de escalonamento nos quais o procedimento de satisfação de restrições é induzido através da utilização iterativa de operadores de reparação de planos de escalonamento aplicados a violações de restrições anteriormente encontradas.

Nos sistemas de Raciocínio baseado em Casos para Escalonamento que fazem reutilização de soluções, é o todo ou parte das soluções dos problemas anteriores que são reutilizados para construir a solução para o novo problema. Um caso contém a descrição do problema e a sua solução, ou parte da solução. Coello e dos Santos (1999) aplicaram este método na resolução de problemas de escalonamento de produção, através de uma *framework*, onde uma solução é reutilizada para resolver novos problemas. Quando não existem casos aplicáveis, o sistema tenta encontrar uma solução através de heurísticas. Oman e Cunningham (2001) usaram este método para a construção de populações iniciais de Algoritmos Genéticos na resolução de problemas de Caixeiro-Viajante e Escalonamento *Job-Shop*. Os autores argumentaram que a definição da população inicial traz vantagens para o processo de pesquisa e que esta vantagem pode ser melhorada com a introdução de informação específica ao problema. Burke *et al.* (2003) aplicaram este método ao escalonamento de horários de cursos universitários, através do desenvolvimento de uma híper-heurística cuja ideia base era manter uma base de casos com a informação das heurísticas bem-sucedidas para uma variedade de problemas anteriores, de modo a prever a melhor heurística para o novo problema. Yang (2004) dissertou sobre o uso deste método para construir soluções para um determinado problema de escalonamento de horários. Neste trabalho, o Raciocínio baseado em Casos era usado para efetuar uma boa seleção de heurísticas iniciais de uma determinada Meta-heurística e demonstrou experimentalmente que uma boa seleção de heurísticas iniciais afeta o desempenho do sistema. Burke *et al.* (2006a) apresentaram uma abordagem que particionava um grande problema de escalonamento de horários em subproblemas, combinava as soluções parciais recuperadas dos subproblemas particionados, e aplicou uma heurística baseada em grafos para construir

a solução para o novo caso. Pereira (2009) e Pereira e Madureira (2013) integraram um módulo de Raciocínio baseado em Casos num Sistema Multiagente de modo a ser possível efetuar uma definição autónoma dos parâmetros de Meta-heurísticas, na resolução de problemas de escalonamento dinâmico.

6.5. Sumário

Neste capítulo foram abordadas várias questões relacionadas com aprendizagem em sistemas baseados em agentes. Foi inicialmente equacionada a problemática da Aprendizagem Automática, nas suas diferentes abordagens. De seguida, foi descrita a relação entre Aprendizagem Automática e Sistemas Multiagente, caracterizando-se os cenários de aprendizagem em agentes e as abordagens de Aprendizagem Multiagente em Equipa e Concorrente

Finalmente, foi equacionada a problemática da aprendizagem na afinação de parâmetros de Meta-heurísticas. Neste sentido, foi dado ênfase às técnicas usadas no âmbito deste trabalho, nomeadamente os algoritmos de *Racing* e o Raciocínio baseado em Casos, tendo sido apresentado, para ambos, a perspetiva histórica, descrição do funcionamento básico e áreas de aplicação.

Capítulo 7. O Sistema Multiagente *AutoDynAgents*

7.1. Introdução

O sistema *AutoDynAgents* (Madureira *et al.*, 2009a; Madureira *et al.*, 2008; Madureira *et al.*, 2009c) é um sistema de escalonamento autónomo no qual uma comunidade de agentes modela um sistema real de fabrico sujeito a perturbações para a resolução distribuída, autónoma e cooperativa de problemas de escalonamento do tipo *Job-Shop* Alargado. É uma extensão das ideias concretizadas no sistema *MASDScheGATS* (Madureira, 2003; Madureira *et al.*, 2007; Madureira *et al.*, 2009b), e foi desenvolvido usando a *framework* JADE[4].

O sistema *AutoDynAgents* consiste num Sistema Multiagente onde existem agentes a representar tarefas e agentes que representam recursos (ou máquinas). Cada Agente Recurso deve ser capaz de encontrar uma solução local quase-ótima (ou mesmo ótima) através da aplicação de uma Meta-heurística implementada (Pesquisa Tabu, *Simulated Annealing*, Algoritmos Genéticos, Otimização por Colónia de Formigas, *Particle Swarm Optimization*, ou Colónia de Abelhas Artificiais), de adaptar os parâmetros de configuração da técnica escolhida de acordo com o problema a resolver, de lidar com o dinamismo no sistema (ou seja, chegada de novas tarefas, cancelamento de tarefas, alteração dos atributos das tarefas, etc.), e comunicar com os outros agentes de modo a resolver o problema de escalonamento de tarefas de produção.

A abordagem de escalonamento seguida pelo *AutoDynAgents* diferencia-se das que são encontradas na literatura, uma vez que neste sistema cada Agente Recurso é responsável por otimizar o escalonamento de operações para a máquina respetiva, através do uso de Meta-heurísticas, e considera um tipo específico de interação social, a Resolução Cooperativa de Problemas, uma vez que um grupo de agentes trabalha conjuntamente para alcançar uma boa solução para o problema (Madureira *et al.*, 2009a; Madureira *et al.*, 2008; Madureira *et al.*, 2009c; Pereira, 2009).

Neste capítulo é descrito o sistema *AutoDynAgents*, nomeadamente a sua arquitetura, módulos e métodos principais. Esta informação baseou-se maioritariamente em (Madureira, 2003; Madureira *et al.*, 2009c; Pereira, 2009; Pereira e Madureira, 2013).

[4] Java Agent DEvelopment Framework (http://jade.tilab.com/)

7.2. Arquitetura global

O problema de escalonamento de tarefas original é decomposto numa série de problemas de máquina única. Os Agentes Recurso têm Meta-heurísticas associadas e obtêm soluções localmente que dão origem a uma solução final viável para o problema global.

De uma forma genérica, o sistema *AutoDynAgents* apresenta de uma arquitetura autónoma com um modelo em equipa e é composto por três componentes principais (Madureira *et al.*, 2009a; Madureira *et al.*, 2008; Madureira *et al.*, 2009c):

1. O módulo de escalonamento híbrido que utiliza Meta-heurísticas e um mecanismo de reparação das atividades entre os recursos. O objetivo deste mecanismo é reparar a operação das máquinas, tendo em consideração as restrições tecnológicas das tarefas, i.e. as relações de precedência das operações, de modo à obtenção de bons planos de escalonamento;
2. O módulo de adaptação dinâmica que inclui mecanismos para regeneração de vizinhanças/populações sob ambientes dinâmicos, aumentando-as ou diminuindo-as de acordo com chegada de novas tarefas ou cancelamento de tarefas existentes;
3. O módulo de coordenação que tem o objetivo de melhorar as soluções encontradas pelos agentes através de cooperação (os agentes agem conjuntamente de modo a melhorar um objetivo comum) ou de negociação (os agentes competem entre si de modo a melhorar as suas soluções individuais).

A arquitetura do sistema *AutoDynAgents* é ilustrada na Figura 7.1 e na Figura 7.2. Enquanto a primeira exibe uma arquitetura global dos vários módulos e sua interligação, a segunda apresenta como os agentes estão interligados entre si. Como é possível verificar, existem seis tipos de agentes: Agente UI, Agentes Tarefa, Agentes Recurso, e os agentes Auto-* (Auto-Configuração, Auto-Otimização, e Auto-Recuperação).

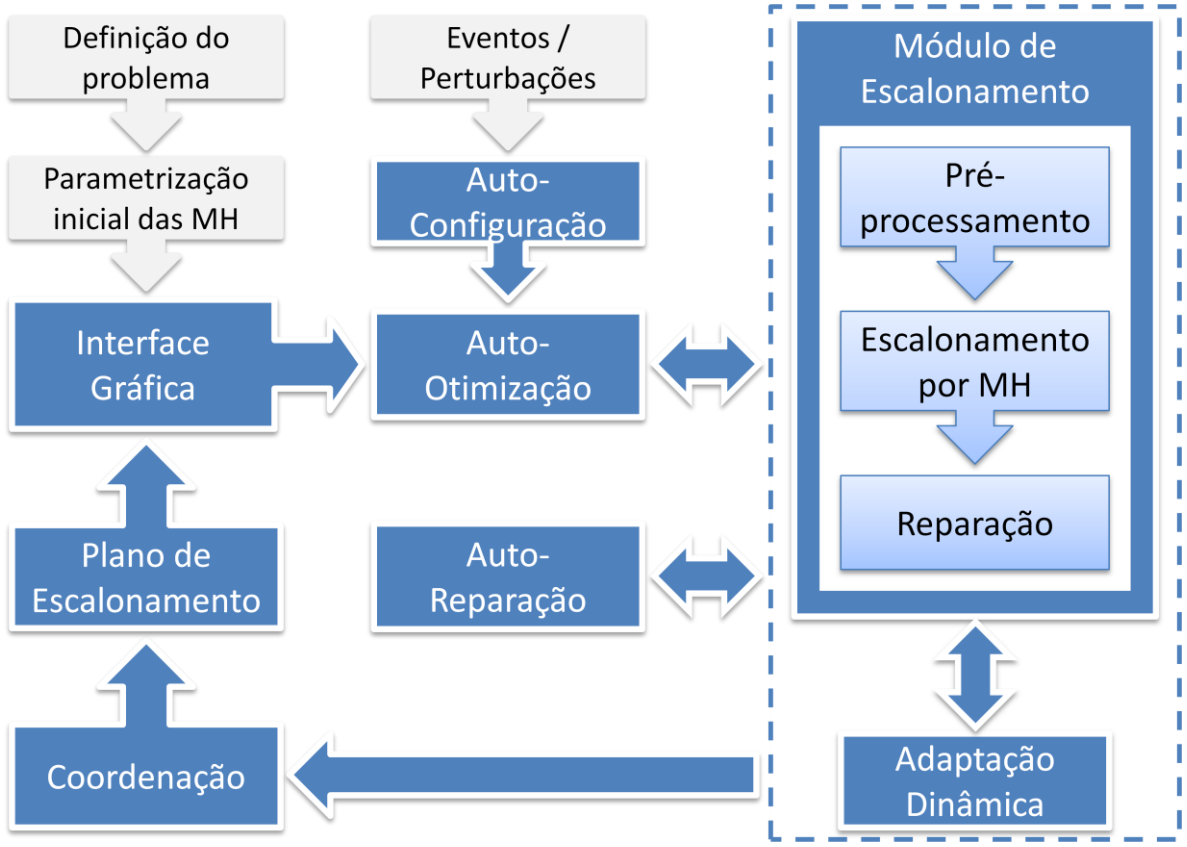

Figura 7.1 – Esquema do sistema *AutoDynAgents*

Da análise da Figura 7.1 é possível identificar a interligação dos vários módulos. O módulo de Interface Gráfica é responsável por identificar a definição do problema e a parametrização base das Meta-heurísticas. O módulo de Auto-Configuração é responsável por detetar alterações no problema (e.g. chegada ou cancelamento de tarefas) originadas pela ocorrência de eventos aleatórios ou perturbações. Estas alterações são detetadas pelo agente e seguidamente comunicadas ao sistema para adaptação dos parâmetros das Meta-heurísticas (através do módulo de Auto-Otimização) e adaptação do problema (através do módulo de Adaptação Dinâmica). O módulo de Auto-Otimização é responsável por selecionar uma Meta-heurística e respetiva parametrização para aplicar à resolução da instância do problema. Por sua vez, o módulo de Auto-Reparação é responsável por detetar falhas no Sistema Multiagente e proceder à respetiva correção. O módulo de Escalonamento (subsecção 7.3), o módulo de Adaptação Dinâmica (subsecção 7.4), e o módulo de Coordenação (subsecção 7.5) geram planos de escalonamento que são apresentados ao utilizador na Interface Gráfica, bem como as medidas de desempenho associadas ao plano.

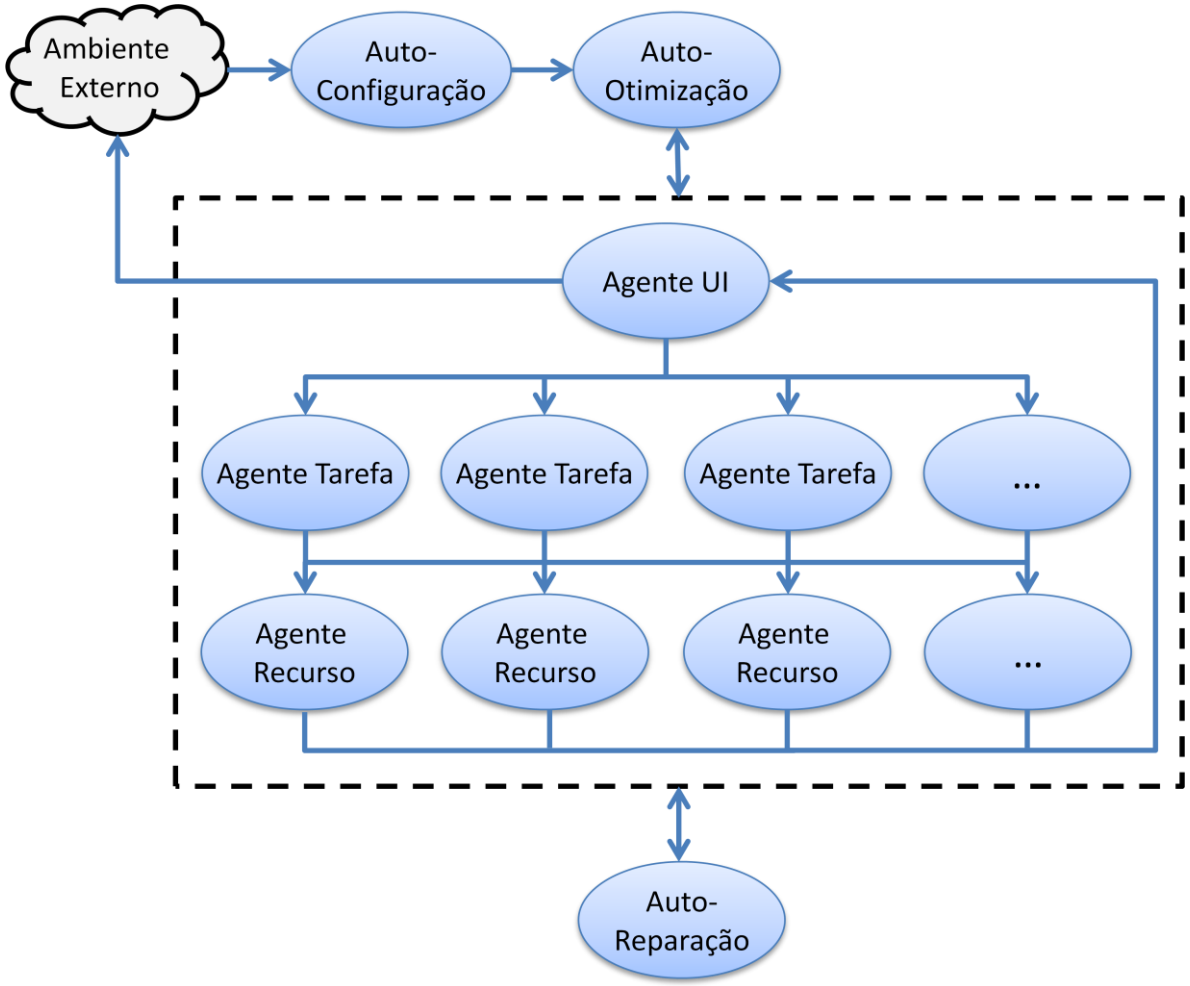

Figura 7.2 – Modelo do sistema *AutoDynAgents* (Pereira, 2009)

Da Figura 7.2 é possível verificar a interligação dos vários agentes presentes na arquitetura, com destaque para a comunicação entre os agentes do módulo de escalonamento. Assim, à semelhança da figura anterior, o agente de Auto-Configuração deteta as alterações ao problema provenientes do ambiente externo (Interface Gráfica ou eventos aleatórios) e comunica-os ao agente de Auto-Otimização, responsável pela definição dos parâmetros de uma Meta-heurística para a resolução do problema. Este por sua vez comunica com o Agente UI, responsável pela criação dos vários Agentes Tarefa, pela coordenação das soluções locais dos Agentes Recurso, e também pela comunicação da solução final ao agente de Auto-Otimização bem como à Interface Gráfica, para a respetiva visualização. Cada Agente Tarefa efetua o pré-processamento da respetiva tarefa e distribui as soluções pelos vários Agente Recurso, responsáveis por obter uma solução local para o problema de máquina única e comunicá-las ao Agente UI. Por último, o agente

de Auto-Reparação encontra-se ligado a todos os agentes do módulo de escalonamento para a deteção e correção de possíveis falhas.

7.2.1. Agente UI

O Agente UI é responsável pela comunicação com o agente de Auto-Otimização e do Sistema Multiagente com o meio exterior, sendo também responsável pela coordenação e reparação das soluções enviadas pelos Agentes Recurso.

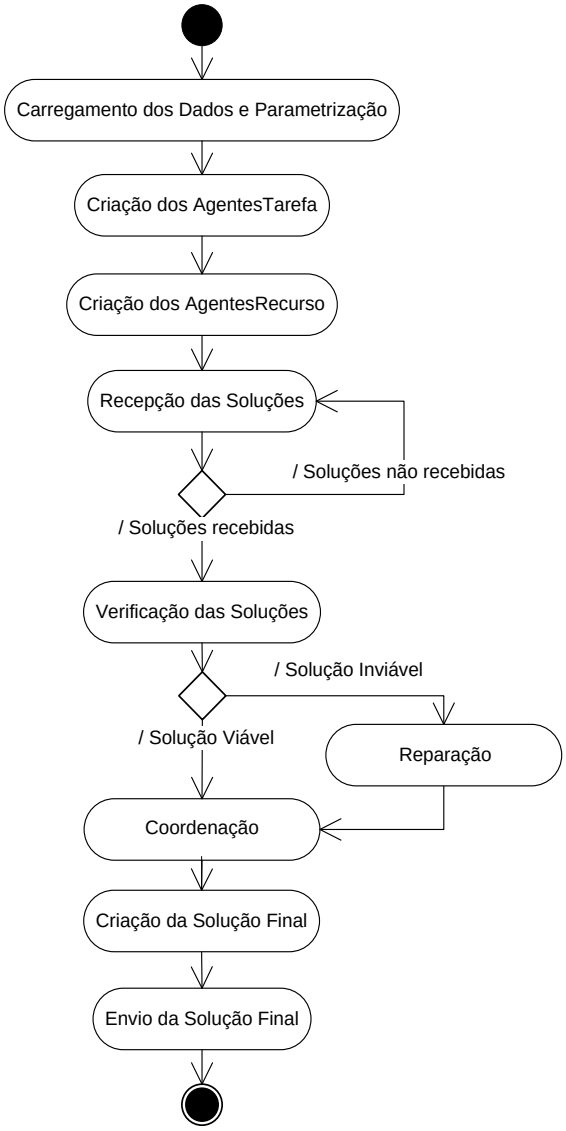

Figura 7.3 – Diagrama de funcionamento do Agente UI

Este agente apresenta as seguintes funcionalidades (Figura 7.3):

- Leitura dos dados relativos às características das tarefas a processar pelo sistema, sejam estas pré-definidas ou dinamicamente construídas, e da parametrização da Meta-heurística a usar (definida pelo agente de Auto-Otimização);
- Criação dinâmica dos Agentes Tarefa (cada agente corresponde a cada tarefa existente no sistema);
- Criação dinâmica dos Agentes Recurso (cada agente corresponde a uma máquina do sistema);
- Recolha das soluções dos vários Agentes Recurso;
- Integração e verificação da exequibilidade das soluções e aplicação do Mecanismo de Reparação no caso das soluções não serem exequíveis;
- Aplicação de um Mecanismo de Coordenação dos agentes, através de Cooperação ou Negociação (opcional);
- Criação da solução final exequível e respetiva comunicação ao meio exterior e agente de Auto-Otimização.

Devido à possibilidade de ocorrência de dinamismo, o Agente UI não termina o seu processamento, ficando à espera de chegada de eventos. No caso de chegada de algum evento, o Agente UI reinicia o processamento, mas só cria os Agentes Tarefa e Agentes Recurso no caso de haver algum novo para criar, e caso a opção de parametrização seja aproveitar o plano anteriormente gerado. No caso de a estratégia passar por recomeçar o escalonamento, então o plano anteriormente gerado é ignorado.

7.2.2. Agentes Tarefa

Os Agentes Tarefa são criados dinamicamente e são únicos por cada tarefa a processar pelo sistema. Cada Agente Tarefa é responsável pelo pré-processamento das operações que constituem a tarefa e pela distribuição e afetação das operações pelos Agentes Recurso. As suas funcionalidades básicas são (Figura 7.4):

- Receção das operações da tarefa (gama operatória), com tempos de processamento, data de conclusão e penalidades associadas;
- Pré-processamento dos dados: cálculo da informação temporal de cada operação (tempos de início e de conclusão previstos);

- Distribuição das operações da tarefa e respetivos atributos pelos respetivos Agentes Recurso.

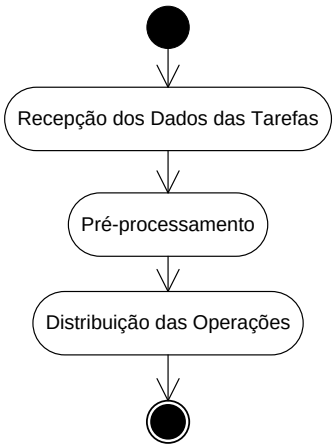

Figura 7.4 – Diagrama de funcionamento dos Agentes Tarefa

No caso de ocorrência de dinamismo, os Agentes Tarefa não terminam o seu processamento após a distribuição das operações. Podem ocorrer alterações nos dados das tarefas que necessitem de re-execução do pré-processamento. No caso de o evento ser de eliminação de uma tarefa, o Agente Tarefa respetivo termina o processamento, comunicando aos Agentes Recurso o sucedido, para que estes possam eliminar as operações respetivas. Quando o novo evento se refere à chegada de tarefas, então novos Agentes Tarefa são criados pelo Agente UI.

7.2.3. Agentes Recurso

Os Agentes Recurso representam as máquinas existentes no sistema para processar as operações constituintes das tarefas. Existe um agente por cada recurso da planta fabril e cada um é responsável pela descoberta da melhor solução de escalonamento das operações recebidas pelos diversos Agentes Tarefa. As funcionalidades deste tipo de agentes são (Figura 7.5):

- Receção das operações e respetivos atributos a serem processadas dos diversos Agentes Tarefa;
- Aplicação de uma Meta-heurística, previamente escolhida e parametrizada pelo agente de Auto-Otimização, para encontrar a melhor solução possível para cada problema de máquina única;
- Comunicação da solução encontrada ao Agente UI.

Estes agentes não terminam o seu processamento ficando à espera da ocorrência de dinamismo para ser reaplicada a Meta-heurística, aproveitando ou não as soluções calculadas anteriormente.

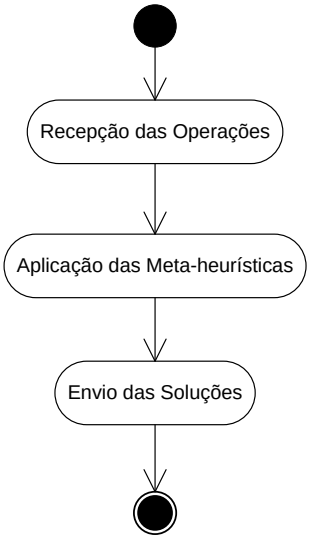

Figura 7.5 – Diagrama de funcionamento dos Agentes Recurso

7.2.4. Agentes Auto-*

Os agentes Auto-* existem para que o conceito de Computação Autónoma seja aplicável ao sistema. As capacidades de autogestão são globais ao sistema, sendo que não existe um módulo único para cada agente do Sistema Multiagente. Assim, cada agente do sistema incorpora os comportamentos de Auto-Configuração, Auto-Otimização e Auto-Recuperação através de comunicações/monitorizações do respetivo agente de autogestão. A capacidade de Auto-Proteção não foi considerada no desenvolvimento por se considerar fora do âmbito deste trabalho.

7.2.4.1. Agente de Auto-Configuração

O agente de Auto-Configuração é responsável por monitorizar o sistema de modo a detetar alterações ocorridas no plano de escalonamento, permitindo ao sistema uma adaptação dinâmica. Com este agente, o sistema está preparado para lidar automaticamente com questões de dinamismo, através da adaptação das soluções a perturbações externas. A adaptação dinâmica é explicada mais detalhadamente na subsecção 7.4.

O agente de Auto-Configuração apresenta as seguintes funcionalidades (Figura 7.6):

- Monitorização contínua do sistema;
- Deteção das perturbações ocorridas;
- Comunicação das perturbações detetadas ao Agente de Auto-Otimização.

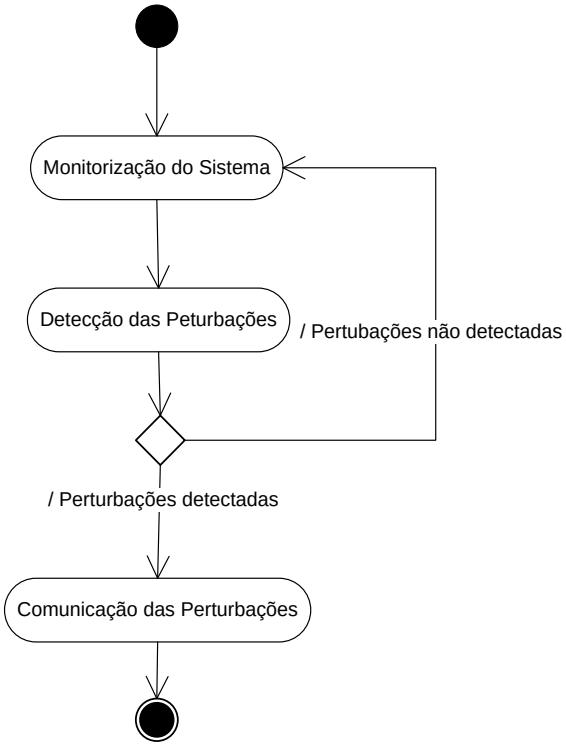

Figura 7.6 – Diagrama de funcionamento do Agente de Auto-Configuração

7.2.4.2. Agente de Auto-Otimização

O agente de Auto-Otimização é responsável pela configuração automática dos parâmetros das Meta-heurísticas, de acordo com a instância do problema a tratar. Como é possível analisar na Figura 7.7, este agente recebe os dados da instância do problema de escalonamento, ou as perturbações detetadas pelo agente de Auto-Configuração, escolhe automaticamente a Meta-heurística a usar, assim como a respetiva parametrização, e comunica com o Agente UI.

Se ocorrer algum tipo de dinamismo, os parâmetros podem ser alterados em tempo de execução. Esta autoparametrização é feita através de aprendizagem e experiência, uma vez que utiliza um módulo de Raciocínio baseado em Casos. Sempre que um novo problema (ou caso) surge, este módulo utiliza a experiência passada de modo a especificar

qual a Meta-heurística e respetivos parâmetros a usar. Quando um novo caso é resolvido, este é armazenado para uso futuro. Esta funcionalidade encontra-se descrita detalhadamente no Capítulo 8, uma vez que se refere a uma das principais contribuições deste trabalho.

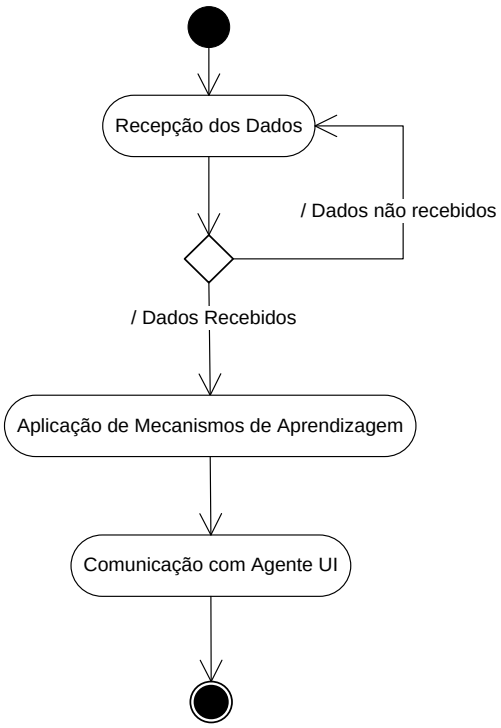

Figura 7.7 – Diagrama de funcionamento do Agente de Auto-Otimização

7.2.4.3. Agente de Auto-Reparação

O agente de Auto-Reparação dá ao sistema a capacidade de diagnosticar desvios às condições normais e despoleta ações proactivamente de modo a normalizar o sistema e a evitar interrupções de serviço. Este agente monitoriza os outros agentes de modo a fornecer capacidades de autorreparação a todos eles (Figura 7.8). Uma vez que os agentes podem falhar por qualquer razão, esta autorreparação permite que os agentes efetuem cópias de segurança dos seus registos, de modo ser a possível a sua reativação para não se perder dados importantes. Os agentes podem ser restaurados a partir de um ponto anterior em vez de ser realizado um *reset* total. Com este agente, o sistema torna-se estável, mesmo se ocorrer alguma falha nos agentes do sistema.

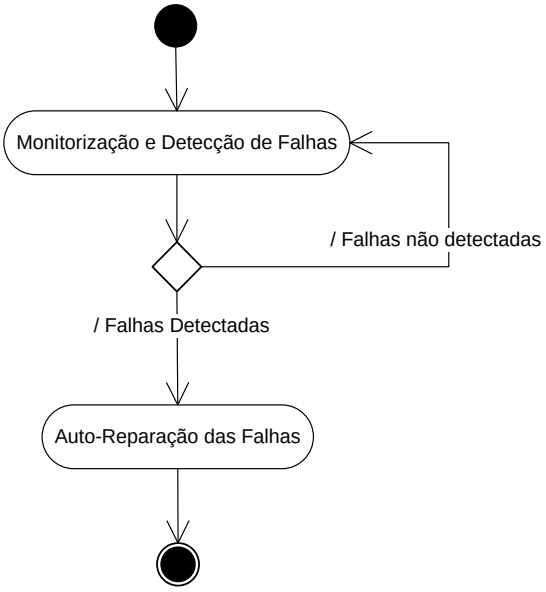

Figura 7.8 – Diagrama de funcionamento do Agente de Auto-Reparação

7.3. Módulo de Escalonamento

O módulo de Escalonamento é constituído pelo Método de Escalonamento baseado em Meta-heurísticas proposto por Madureira (2003). Na Tabela 7.1 é apresentada a notação usada na descrição dos módulos e métodos.

Tabela 7.1 – Notação do módulo de Escalonamento (Madureira, 2003)

m – Número de máquinas
n – Número de tarefas
l – Nível da operação definido no grafo de precedências
O_{ijkl} – Operação k da tarefa j, a ser processada na máquina i com o nível l
IO_{ijkl} – Intervalo de tempo para iniciar a operação O_{ijkl}
d_j – Data de entrega para a tarefa j
t_j – Tempo de início de processamento para a tarefa j
r_j – Tempo de lançamento da tarefa j
r_{ijkl} – Tempo de lançamento da operação O_{ijkl}
t_{ijkl} – Tempo mais cedo no qual a operação O_{ijkl} pode começar
T_{ijkl} – Tempo mais tarde no qual a operação O_{ijkl} pode começar
p_{ijkl} – Tempo de processamento da operação O_{ijkl}
C_{ijkl} – Tempo de conclusão da operação k da tarefa j, nível l na máquina i
L_j – Atraso da tarefa ($L_j = C_j - d_j$)
T_j – Atraso efetivo (*tardiness)* da tarefa ($T_j = \max \{ L_j , 0 \}$)

As soluções para os diferentes problemas de máquina única são codificadas através de representação direta, onde o plano de escalonamento é descrito como uma sequência de operações, i.e. 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 (o nível 1 corresponde às operações iniciais, sem precedentes).

7.3.1. Funcionamento geral

O Método de Escalonamento baseado em Meta-heurísticas, como descrito na Tabela 7.2, é implementado em duas fases (Madureira, 2003):

1. A primeira fase tem como objetivo obter um plano para o problema *Job-Shop* Alargado, baseado na decomposição do problema original nos vários problemas de máquina única que o constituem. Estes problemas de máquina única são resolvidos por cada Agente Recurso. Nesta fase é realizado um pré-processamento inicial, i.e. cada Agente Tarefa usa diferentes mecanismos para a definição do intervalo dos tempos estimados de início e dos tempos de conclusão das operações de cada tarefa. De seguida é aplicada uma Meta-heurística a cada um dos problemas de máquina única (associados a um Agente Recurso), e integradas as soluções obtidas no problema principal. Esta fase tem já como resultado um plano de escalonamento (possivelmente não exequível);
2. A segunda fase tem a finalidade de verificar se o plano de escalonamento obtido é exequível. Se o plano não for válido, é aplicado o Mecanismo de Reparação que redefine os tempos de início e de conclusão para cada operação, de modo a ser obtido um plano de escalonamento exequível (descrito com mais detalhe na subsecção 7.3.2).

Inicialmente, o problema *Job-Shop* Alargado determinístico é decomposto em problemas de máquina única. Assume-se a existência de datas de lançamento r_j diferentes e conhecidos das tarefas, bem como datas de entrega d_j. Com base nestes tempos, são determinadas as datas de lançamento e de entrega para cada problema de máquina única, sendo cada um desses problemas resolvido no Agente Recurso respetivo através da aplicação de uma Meta-heurística. Seguidamente, as soluções obtidas para cada problema de máquina única são integradas de modo a se obter uma solução para o problema inicial.

Tabela 7.2 – Método de Escalonamento baseado em Meta-heurísticas (Madureira, 2003)

1ª Fase	**Encontrar um plano para o problema *Job-Shop* Alargado, 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.
2ª Fase	**Verificar a exequibilidade do plano obtido e repará-lo se necessário.**
Passo 6	Verificar se estamos perante uma solução válida ou exequível. Se não for válida então é necessário aplicar o Mecanismo de Reparação.

O tempo de conclusão para cada operação é derivado das datas de entrega das tarefas respetivas e do tempo de processamento através da diferença do tempo de processamento ao tempo de conclusão da tarefa imediatamente seguinte (equação (7.1))

$$C_{ij(k-1)l} = C_{ijkl} - p_{ijkl} \tag{7.1}$$

Este procedimento começa com a última operação da tarefa e termina na primeira. Quando uma operação é precedida por mais do que uma operação, i.e. quando existe uma estrutura multinível, o tempo de conclusão é o valor mais baixo, como definido na equação (7.2).

$$C_{ijk(l-1)} = \min\{C_{ijkl} - p_{ijkl}\} \ \forall\, l > l-1 \tag{7.2}$$

Os intervalos de tempo de início das operações [t_{ijkl},T_{ijkl}] são também definidos considerando os tempos de lançamento das tarefas e os tempos de processamento das operações. O tempo de início mais cedo t_{ijkl} corresponde ao instante de tempo a partir do qual o processamento da operação pode ter início. O tempo de início mais tarde T_{ijkl} corresponde ao tempo para o qual o processamento da operação deve começar, de modo a cumprir a data de entrega da tarefa. Quando uma operação tem mais do que uma operação precedente, o intervalo [t_{ijkl},T_{ijkl}] é a interceção dos intervalos das operações precedentes

correlacionados pelos tempos de processamento respetivos. Nesta fase, apenas são consideradas as restrições de precedência tecnológica das operações e as datas de entrega das tarefas, para definição dos tempos de início e de conclusão.

O intervalo de tempo de início (ITI) para operações sem precedentes é definido como descrito na equação (7.3).

$$ITI_{ijkl} = [r_{ijkl}, C_{ijkl} - p_{ijkl}] \tag{7.3}$$

O ITI para operações com uma precedente é definido na equação (7.4).

$$ITI_{ijkl} = \left[t_{ijk(l-1)}, T_{ijk(l-1)}\right] + p_{ijk(l-1)} \tag{7.4}$$

O intervalo de tempo de início para operações com mais do que uma precedente é a interceção dos intervalos de tempo de início de todas as operações precedentes correlacionadas pelos respetivos tempos de processamento (equação (7.5)).

$$\begin{aligned} ITI_{ijkl} = \left[t_{ijK1L} + p_{ijK1L}, T_{ijK1L} + p_{ijK1L}\right] \cap \left[t_{ijK2L} + p_{ijK2L}, T_{ijK2L} + p_{ijK2L}\right] \cap \ldots \\ \cap \left[t_{ijKnL} + p_{ijKnL}, T_{ijKnL} + p_{ijKnL}\right] com\ Kn < k\ e\ L < l \end{aligned} \tag{7.5}$$

7.3.2. Implementação das Meta-heurísticas

Nesta subsecção são descritos alguns detalhes da implementação das várias Meta-heurísticas. A Pesquisa Tabu e os Algoritmos Genéticos foram importados do sistema *MASDScheGATS*.

Na implementação da Pesquisa Tabu e dos Algoritmos Genéticos foi considerado o conceito de afastamento máximo na geração de vizinhanças/populações (Madureira, 2003). Uma vez que se utiliza permutações de tarefas, é possível definir um afastamento máximo para que se possa restringir algumas soluções que podem levar a planos não exequíveis. Por exemplo, poderá não fazer sentido trocar a primeira tarefa com a última. Assim sendo, com o conceito de afastamento máximo, o espaço de soluções é menor, o que possibilita uma maior eficiência na procura de soluções. O afastamento máximo é representado através de uma percentagem do número de tarefas do problema (e.g., com um afastamento

máximo de 25% num problema de 20 tarefas, cada tarefa poderá trocar com um dos seus 5 vizinhos à direita ou à esquerda).

Foram também consideradas algumas das regras de prioridade descritas na secção 2.4, para a geração da solução inicial, nomeadamente a EDD, SPT, REDD, RND, e *SeqNivel*.

7.3.2.1. Pesquisa Tabu

A implementação da Pesquisa Tabu foi desenvolvida com recurso a uma *framework* desenvolvida em Java por Robert Harder, designada *OpenTS*[5], que suporta o paradigma de programação orientada a objetos.

Uma vez que esta *framework* foi desenvolvida para funcionar para qualquer tipo de problemas, foi necessária a implementação de algumas classes, nomeadamente para lidar com a estrutura das soluções, para calcular a função objetivo, para efetuar o movimento de troca de soluções, e para gerar todos os movimentos possíveis em cada iteração. Nesta última classe foi implementado o mecanismo de geração de vizinhanças anteriormente descrito.

A utilização desta *framework* revelou algumas vantagens pois segue o algoritmo original e, além de ser eficaz e eficiente, lida automaticamente com alguns mecanismos próprios do algoritmo, salientando-se o facto de percorrer a vizinhança de soluções, selecionar a melhor solução, e gerir a lista tabu, duma forma transparente para o programador.

Os parâmetros desta Meta-heurística são:

- **Critério de Paragem** – número de iterações que o algoritmo efetua;
- **Afastamento máximo** – percentagem de geração de vizinhança, dependendo do tamanho do problema;
- **Subvizinhança** – percentagem do tamanho da vizinhança usada ao longo da execução;
- **Lista Tabu** – tamanho da lista tabu.

[5] http://www.coin-or.org/Ots/index.html

7.3.2.2. *Simulated Annealing*

Esta Meta-heurística foi implementada diretamente a partir do algoritmo original, descrito na subsecção 3.2.3. Os parâmetros do algoritmo são:

- **Critério de Paragem** – número de iterações que o algoritmo efetua;
- **Número de iterações *k*** – número de iterações à mesma temperatura;
- **Temperatura Inicial** – temperatura com que o algoritmo começa a processar;
- ***Alpha*** – representa o fator de redução de temperatura, definido no intervalo]0-1[.

A única alteração realizada no algoritmo de modo a aumentar a eficiência foi utilizar um método de geração de vizinhança ligeiramente diferente. Normalmente é gerada a vizinhança sendo depois escolhido um vizinho aleatoriamente. No entanto, é necessário gerar e guardar a totalidade da vizinhança em todas as iterações, o que se revela pouco eficiente. Para tentar incrementar a eficiência, considerou-se uma seleção aleatória do índice do vizinho a selecionar antes da geração da vizinhança, para que o método retorne quando encontrar o vizinho selecionado, em vez de se gerar a totalidade da vizinhança e só depois selecionar um vizinho aleatoriamente. Desta forma não é necessário guardar a vizinhança e, apenas no pior caso, é percorrida a totalidade da vizinhança, melhorando a eficiência. De modo a ser possível saber o número total de vizinhos, para proceder à seleção aleatória de um vizinho, utilizou-se a fórmula da equação (7.6).

$$num_{vizinhos} = k(n-k) + \sum_{i=n-k+1}^{n-1} n - i \qquad (7.6)$$

com *n* igual ao número de tarefas e *k* igual ao número máximo de posições a trocar.

7.3.2.3. Algoritmos Genéticos

Esta Meta-heurística foi implementada com recurso a uma *framework* externa, a *WATCHMAKER*[6], desenvolvida por Daniel W. Dyer. Esta é uma *framework* orientada a objetos para Computação Evolucionária, desenvolvida em Java.

Assim como na *framework* da Pesquisa Tabu, também esta necessitou do desenvolvimento de algumas classes específicas ao problema de escalonamento, nomeadamente para cálculo da função objetivo, para geração de populações (onde se

[6] http://watchmaker.uncommons.org/

implementou o mecanismo de geração de populações descrito anteriormente), e para cada operador usado (cruzamento, mutação e seleção).

O algoritmo é seguido com total transparência para o programador, e sem que este necessite de se preocupar com o desenvolvimento, evoluindo automaticamente, e retornando o melhor indivíduo da população.

Os parâmetros dos Algoritmos Genéticos são:

- **Número de gerações** – número de gerações que o algoritmo realiza;
- **Geração da população inicial** – percentagem de geração da população inicial, vulgo afastamento máximo;
- **Tamanho da população** – percentagem do tamanho da população usada ao longo da execução;
- **Taxa de cruzamento** – taxa de cruzamento de indivíduos;
- **Taxa de mutação** – taxa de mutação de indivíduos.

7.3.2.4. Otimização por Colónia de Formigas

Na implementação desta Meta-heurística tentou usar-se uma aplicação o mais direta possível do algoritmo original, descrito na subsecção 3.2.5, embora com algumas adaptações. Considera-se um caminho composto por vários ramos, sendo cada um desses ramos um par de tarefas (por exemplo, A->B, E->C, etc.).

Os parâmetros de entrada são:

- **Número de iterações** – número de iterações máximo por colónia;
- **Número de formigas** – número de formigas por colónia;
- **Número de colónias** – número de colónias existentes. Foi dado suporte para múltiplas colónias;
- **Taxa de evaporação** – taxa de evaporação de feromona;
- ***Alpha*** – importância do valor heurístico;
- ***Beta*** – importância da feromona.

Foi dado suporte para múltiplas colónias, tendo cada colónia a sua lista de formigas. Cada formiga tem um caminho atual a percorrer e uma lista de caminhos já percorridos. Sempre que tenta mudar de caminho, cada formiga verifica se o caminho já está na lista e só permite a mudança se este ainda não estiver incluído na mesma. Cada caminho tem uma

solução de tarefas e o respetivo valor objetivo, sendo a construção dos caminhos feita ramo a ramo. Cada colónia estagna quando todas as formigas estão a percorrer o mesmo caminho.

Em cada colónia existe uma matriz que guarda a feromona dos ramos e em cada célula desta matriz é armazenada a feromona existente no ramo *i->j*. Uma particularidade da matriz é o facto dos campos em que *i=j*, ou seja, as diagonais da matriz, corresponderem ao caso de um nó *i* ser a primeira tarefa da solução. Inicialmente todos os campos da matriz são inicializados a -1.

Para uma melhor compreensão, considere-se, a título de exemplo, duas formigas num problema de 5 tarefas, cada uma a percorrer um caminho diferente (BCEAD e ECBDA). Cada formiga deposita a feromona nos ramos, de forma a depositar tanto mais feromona quanto melhor for a qualidade do caminho.

Tabela 7.3 – Exemplo de matriz de feromona

i \ j	A	B	C	D	E
A	-1	-1	-1	0.5	-1
B	-1	0.5	0.5	0.2	-1
C	-1	0.2	-1	-1	0.5
D	0.2	-1	-1	-1	-1
E	0.5	-1	0.2	-1	0.2

Na Tabela 7.3 é apresentada a matriz de feromona no fim de uma iteração. É possível verificar claramente as diferenças de feromona nos ramos, sendo a feromona de um caminho igual a 0.2 e a feromona do outro igual a 0.5.

Em cada iteração, enquanto as formigas constroem os caminhos, a escolha de cada ramo funciona como uma probabilidade, i.e. poderá ser escolhido um ramo que deteriora a solução, não sendo sempre garantida uma escolha dos melhores ramos, embora estes tenham, obviamente, uma maior probabilidade de serem escolhidos.

O valor heurístico da fórmula de escolha dos caminhos (equação (3.3), página 36) é calculado em tempo real, na construção de um caminho, ou seja, vai sendo aplicada a função objetivo na construção iterativa de um caminho. Isto permite saber o quanto um caminho é bom em dado momento. Foi usada esta abordagem devido ao facto da função objetivo poder variar consoante a parametrização escolhida.

Depois de um novo caminho estar construído, este é comparado com o caminho atual que a formiga está a percorrer e só é feita uma mudança de caminho se o novo caminho for melhor que o atual. Esta restrição foi colocada para não permitir que a formiga esteja, no pior caso, sempre a mudar de caminho, impedindo a estagnação da colónia, evitando-se que uma formiga esteja a percorrer um bom caminho e mude para um pior. Com base nos testes efetuados, foi possível verificar que os resultados obtidos melhoram sensivelmente.

No final, quando uma colónia estagna ou quando atinge o número máximo de iterações, o caminho com mais feromona é devolvido.

7.3.2.5. *Particle Swarm Optimization*

À semelhança das outras Meta-heurísticas, também o *Particle Swarm Optimization* foi implementado tentando ser o mais fiel possível ao algoritmo inicial. No entanto, foram introduzidas algumas adaptações.

O *Particle Swarm Optimization* é um algoritmo bastante simples e com toda a lógica do seu funcionamento nas suas funções de cálculo de velocidade e posição para cada partícula. Na aplicação a problemas de Escalonamento, essa sua simplicidade de compreensão na implementação e funcionamento é agravada, podendo até tornar-se um pouco confusa.

Inicialmente o algoritmo foi pensado para lidar com um certo número de dimensões lógicas. Cada uma dessas dimensões diz respeito a uma coordenada num referencial independente. Com o conjunto dessas coordenadas tem-se a posição da partícula em cada um desses referenciais (e.g. um referencial de 3 dimensões corresponde a um referencial *xyz*).

Assim sendo, o algoritmo é constituído por partículas, cada uma com as suas dimensões, por uma melhor solução encontrada (ótimo local), e por um valor de *fitness* atual da partícula. Considerando que esse conjunto de partículas diz respeito a uma população, existe uma forma de guardar a melhor solução encontrada desde o início de funcionamento do algoritmo (melhor solução global).

Juntando estes componentes, não se encontra uma forma fácil e com lógica para adaptá-lo ao problema de escalonamento. A forma encontrada e usada para essa adaptação passa por fazer corresponder as dimensões ao número de tarefas. E é aqui que

tudo fica mais complexo. Ou seja, se a cada dimensão corresponder uma tarefa, e se considerarmos um problema com 40 tarefas, iremos colocar o algoritmo em funcionamento com *N* partículas, contendo cada uma, 40 dimensões.

Os cálculos de velocidade e posicionamento são efetuados para cada uma dessas dimensões de cada partícula. O seu valor de *fitness* é calculado após ordenar as dimensões por ordem crescente (minimização) e guardado no *fitness* atual de cada uma. O valor do ótimo local será substituído sempre que a ordenação das dimensões dessa partícula resultar num valor de *fitness* com mais qualidade que o anterior.

Relativamente ao valor da melhor solução global, como descrito na literatura, este é obtido após apuramento da solução com mais qualidade encontrada entre os ótimos locais. Foi preciso ter em atenção que, quando as dimensões de cada partícula são ordenadas para cálculo do *fitness*, essa ordenação é temporária, efetuada apenas para esse cálculo.

Utilizou-se uma classe responsável por armazenar a informação referente a cada partícula, nomeadamente o ótimo local, o *fitness* da solução, o *fitness* da melhor solução global, e a estrutura de dados referente aos valores para cada dimensão. Nessa estrutura de dados para as dimensões, é possível guardar os atributos relativos às tarefas, bem como a velocidade e a posição. Embora a velocidade e posição não façam parte da informação de entrada do problema a tratar, foram incluídos em conjunto com os atributos de entrada das tarefas, visto que o seu nível de dependência e hierarquia em relação às partículas e às suas dimensões era igual.

Os parâmetros desta Meta-heurística são:

- **Número de iterações** – critério de paragem do algoritmo;
- **Número de partículas** – número de partículas;
- **Velocidade mínima** – velocidade mínima das partículas;
- **Velocidade máxima** – velocidade máxima das partículas;
- **Inércia mínima** – inércia mínima usada pelo algoritmo;
- **Inércia máxima** – inércia máxima usada pelo algoritmo;
- **C1** – componente cognitiva;
- **C2** – componente social;
- **Limite inferior** – limite inferior da partícula usado pelo algoritmo;
- **Limite superior** – limite superior da partícula usado pelo algoritmo.

7.3.2.6. Colónia de Abelhas Artificiais

Na implementação desta Meta-heurística usou-se um algoritmo ligeiramente modificado, pois o algoritmo original foi proposto inicialmente para problemas sem restrições. Esta versão modificada foi proposta por (Karaboga e Akay, 2011) para obter melhor desempenho em problemas de otimização com restrições, que é o caso do problema de escalonamento.

De forma geral, as principais mudanças no algoritmo consistiram em incorporar mais parâmetros de controlo de modo a melhorar a sua capacidade de convergência para problemas de otimização com restrições. Outra alteração passou pelo uso de uma abordagem de seleção baseada nas regras de seleção de Deb (2000) em vez de uma seleção gananciosa. Para além disso, por permitir soluções inviáveis na população, de modo a garantir alguma diversidade, o algoritmo atribui valores probabilísticos às soluções inviáveis que são inversamente proporcionais às suas violações de restrições, o que lhes dá hipóteses de serem selecionadas no processo de seleção de abelhas espectadoras (Karaboga e Akay, 2011).

Através da aplicação das regras de seleção de Deb (2000), as abelhas mantêm a posição ou memorizam a nova posição através do esquecimento da posição antiga. O método de Deb (2000) utiliza um operador de seleção por torneios, onde duas soluções são comparadas em cada instante através da aplicação dos seguintes critérios:

- Qualquer solução viável é preferida a qualquer solução inviável;
- Entre duas soluções viáveis, é preferida aquela que tem melhor valor na função objetivo;
- Entre duas soluções inviáveis, é preferida aquela que viola menos restrições;

Com base nestes critérios, as probabilidades para as abelhas espectadoras são calculadas através da equação (7.7), que substitui a equação (3.7) da página 43.

$$P_i = \begin{cases} 0.5 + \left(\dfrac{fitness_i}{\sum_{j=1}^{sn} fitness_j}\right) \times 0.5, se\ a\ solução\ for\ viável \\ \left(1 - \dfrac{violação_i}{\sum_{j=1}^{sn} violação_j}\right) \times 0.5, se\ a\ solução\ for\ inviável \end{cases} \quad (7.7)$$

onde $violação_i$ representa a penalização da solução *i* e $fitness_i$ representa o valor da função objetivo da solução *i,* o qual é proporcional à quantidade de néctar da fonte de comida. O valor de *fitness* é determinado pela equação (7.8).

$$fitness_i = \begin{cases} \frac{1}{1+f_i}, se\ f_i \geq 0 \\ 1 + abs(f_i), se\ f_i < 0 \end{cases} \quad (7.8)$$

onde f_i é o valor de custo da solução *i.*

Os parâmetros desta Meta-heurística são:

- **Tamanho da população** – número de abelhas na colónia;
- **Insucesso máximo** – número máximo de ciclos permitidos para tentar melhorar uma solução;
- **Número de ciclos** – critério de paragem do algoritmo;

7.3.3. Mecanismo de Reparação

A integração das soluções para os vários problemas de máquina única pode levar a planos inviáveis para o problema de escalonamento em questão. Tal acontece por se considerar em cada máquina apenas as relações de precedência nas máquinas e as datas de conclusão das tarefas. A atividade inter-relacionada das várias máquinas não é tida em consideração no momento da resolução de cada problema de máquina única, logo a coordenação das soluções tem de ser feita *a posteriori.*

O Mecanismo de Reparação descrito na Tabela 7.4, tem o objetivo de reparar a solução obtida, através da coordenação dos tempos de ocupação das máquinas e das relações de precedência entre operações, redefinindo ou confirmando os tempos de início e de conclusão para cada operação de modo a ser obtido um plano de escalonamento exequível (Madureira, 2003).

Tabela 7.4 – Mecanismo de Reparação (Madureira, 2003)

Passo 1	Reparar a atividade das máquinas começando pelas operações de primeiro nível, uma vez que estas não têm operações precedentes. Neste nível, os instantes de início e de conclusão das operações mantêm-se, i.e. são iguais aos definidos pelo algoritmo de escalonamento na fase anterior.
Passo 2	A seguir consideramos todas as operações cujas precedentes já tenham sido escalonadas. $$t_{ijkl} \leftarrow \max(C_j, Tconcl_j)$$ em que $Tconcl_j$ é o instante de conclusão da operação precedente na sequência para processamento na máquina e C_j é o instante de conclusão da operação precedente da mesma tarefa.
Passo 3	O passo 2 repete-se até todas as operações terem sido escalonadas.

O plano resultante é constituído por um conjunto de sequências de operações, uma por cada máquina. O processo de reparação consiste em percorrer todas as sequências e tentar "coordenar" a atividade das máquinas, tendo em conta as restrições de precedência definidas para cada uma das tarefas e os tempos de ocupação das máquinas onde estas irão ser processadas.

Para cada uma das operações é necessário estabelecer ou confirmar os instantes de início e de conclusão de cada operação. O tempo de início t_{ijkl} de cada operação é igual ao maior dos seguintes valores:

- Instante de conclusão da operação imediatamente precedente da mesma tarefa, no caso, de haver mais do que uma, corresponde ao maior dos tempos de conclusão;
- Instante de conclusão da operação que lhe precede na máquina.

7.4. Módulo de Adaptação Dinâmica

Em problemas não-determinísticos alguns parâmetros são incertos, i.e. não são fixos como se assume nos problemas determinísticos. O não-determinismo de variáveis deve ser tido em consideração na resolução de problemas de mundo-real. Para a geração de soluções aceitáveis em tais circunstâncias, o sistema *AutoDynAgents* começa por gerar um plano preditivo[7] e depois, se ocorrer alguma perturbação durante a execução, o plano de escalonamento pode ser alterado ou revisto, ou seja, é efetuado um reescalonamento.

Considerando os tempos de processamento envolvidos e a grande frequência de perturbações, o reescalonamento de todas as tarefas desde o início deve ser evitado. No entanto, se o processamento de tarefas ainda não começou, pode-se reescalonar desde o início, já com a perturbação incluída no problema. Quando não é viável reescalonar desde o início, ou se o processamento de tarefas já começou, deve ser usada uma estratégia que adapte o plano de escalonamento atual tendo em consideração o tipo de perturbações ocorrido.

O Módulo de Adaptação Dinâmica (Madureira *et al.*, 2011a) tem o objetivo de analisar os eventos que ocorrem e permitir que sistema se adapte a estes.

[7] Plano de escalonamento global gerado com antecedência

Os tipos de eventos que requerem reescalonamento são (Madureira, 2003):

- **Eventos parciais**, que implicam alterações nos atributos das tarefas ou operações, tais como tempos de processamento, datas de entrega, datas de lançamento ou pesos de tarefas;
- **Eventos globais**, que implicam alterações na estrutura das vizinhanças, como resultado de chegada ou cancelamento de tarefas.

Na ocorrência de um evento global, i.e. se uma nova tarefa é introduzida ou cancelada no sistema, a estrutura das soluções/cromossomas deverá modificar-se, aumentando ou diminuindo o tamanho destas, pelo aumento ou diminuição do número de posições da sequência e posterior reavaliação, havendo também uma alteração da estrutura da vizinhança/população, pelo aumento ou diminuição do número de soluções/indivíduos. Quando uma nova tarefa é introduzida no sistema, é necessário aplicar um Mecanismo de Integração, que consiste na introdução da nova tarefa na solução atual. O Mecanismo de Integração analisa o grafo de precedências de operações e, de acordo com a regra respetiva, introduz cada operação da tarefa no problema de máquina única respetivo.

No sistema *AutoDynAgents* são considerados os seguintes Mecanismos de Integração, conforme trabalho prévio (Madureira, 2003):

- **Aleatório (RND)** – consiste em selecionar aleatoriamente uma posição e inserir a nova tarefa nessa posição na solução atual ou em todos os cromossomas;
- **Regra DEMC (Data de Entrega Mais Cedo)** – consiste em inserir a nova tarefa antes da primeira posição possível que possua uma data de entrega posterior;
- **Regra DLMC (Data de Lançamento Mais Cedo)** – consiste em inserir a nova tarefa antes da primeira posição possível que possua uma data de lançamento superior;
- **Regra MP (Maior Prioridade)** – consistem em inserir a nova tarefa antes da primeira posição possível que possua uma prioridade inferior;
- **Regra FIFO (*First In First Out*)** – as tarefas são colocadas no fim das restantes à medida que vão chegando ao sistema.

Quando uma tarefa é cancelada, é necessário aplicar um mecanismo de eliminação, para que as operações correspondentes sejam eliminadas da solução atual em todos os Agentes Recurso.

Depois de resolvido o problema de integração ou eliminação de tarefas é necessário implementar um mecanismo de regeneração que tem por finalidade restaurar a vizinhança ou população modificada de forma a ser assegurada uma estrutura idêntica. A chegada de uma nova tarefa requer a determinação dos tempos de início e de conclusão de cada operação da nova tarefa, e posterior aplicação de um mecanismo de integração, como já descrito anteriormente. A eliminação de uma tarefa requer a aplicação de um mecanismo de eliminação que retire ou elimine a posição correspondente, para a eliminação das operações nos problemas das máquinas respetivas.

A alteração do tempo de processamento de uma operação implica a alteração dos tempos de início e de conclusão das operações que lhe sucedem no grafo de precedências da tarefa a que pertencem. A alteração da data de entrega de uma tarefa implica a alteração dos tempos de início e de conclusão das operações que a constituem. A modificação da data de lançamento de uma tarefa implica a adaptação consequente dos intervalos dos tempos de início das operações que a constituem.

O dinamismo pode ser considerado de duas formas no sistema *AutoDynAgents*:

1. Existe um plano de escalonamento preditivo, gerado pelo Módulo de Escalonamento, sendo este reescalonado ou adaptado dinamicamente, sempre que existe uma ocorrência externa. Para tal são usados os Mecanismos de Integração e Regeneração descritos anteriormente;
2. Surge uma perturbação e o Módulo de Escalonamento ainda está a reconstruir o plano anterior, em resultado de anteriores perturbações.

Na primeira situação, considera-se a revisão ou adaptação do plano atual ao nível do Módulo de Adaptação Dinâmica e posterior refinamento no Módulo de Escalonamento. Na segunda, estamos em pleno processo de geração do novo plano, usando uma qualquer Meta-heurística, surge a necessidade de introduzir o novo evento na solução ou população atual e continuação do processo de pesquisa de melhores soluções.

O Agente UI recebe os eventos ocorridos detetados pelo agente de Auto-Configuração, interpreta-os e comunica ao Agente Tarefa correspondente. Se o evento é de chegada, um novo Agente Tarefa é criado, comunicando aos Agentes Recurso as novas operações. Se o evento é de eliminação de tarefa, o Agente Tarefa informa os Agentes Recurso para estes eliminarem da solução as operações correspondentes. Se for de alteração dos dados das tarefas, o Agente Tarefa efetua as alterações necessárias.

7.5. Módulo de Coordenação

Ao longo do desenvolvimento deste trabalho, e em paralelo, foram desenvolvidas duas abordagens de coordenação de agentes, descritas aqui para completar a descrição do sistema *AutoDynAgents*. A primeira segue uma filosofia de cooperação e a segunda adota uma vertente competitiva.

7.5.1. Mecanismo de Cooperação

O Mecanismo de Cooperação (Madureira *et al.*, 2014; Madureira *et al.*, 2013a; Madureira *et al.*, 2010a) tem o propósito de incorporar comportamentos de inteligência cooperativa no sistema *AutoDynAgents*, de modo a que seja possível analisar e melhorar o plano de escalonamento gerado pelos Agentes Recurso. O principal objetivo é reduzir os tempos mortos do plano gerado e ao mesmo tempo reduzir o tempo de conclusão do plano e maximizar a taxa de utilização do sistema.

O conceito principal deste mecanismo passa por minimizar os tempos mortos dos recursos através da troca de operações, mas respeitando as restrições impostas pelo problema. Os tempos mortos nos Agentes Recurso são gerados pelas restrições de precedência das operações, i.e. uma operação tem de esperar que a anterior seja processada, quer na máquina quer na respetiva tarefa.

Algoritmo 7.1 – Mecanismo de Cooperação (Madureira *et al.*, 2014)

```
    Entrada: ρ  // Plano resultante do Mecanismo de Reparação
    Saída: ρ // Melhor plano obtido
1   v ← getTemposMortos(ρ); // Obtém os tempos mortos ordenados por ordem decrescente
2   Enquanto tamanho(v) > 0 Faz // Enquanto existirem tempos mortos por tratar
3           Se testaTrocaOperacoesSucedentes(ρ,v) Então
4                   // Troca a operação pela seguinte, se possível
5                   ρ ←trocaOperacoesSucedentes(ρ,v);
6           Senão
7                   Se testaTrocaOperacoesPrecedentes(ρ,v) Então
8                           // Troca a operação pela sua precedente, se possível
9                           ρ ←trocaOperacoesPrecedentes(ρ,v);
10                  Fim
11          Fim
12          v .removeTempoMorto (); // Remove o tempo morto da lista, pois já foi tratado
13  Fim
14  Devolver ρ;
```

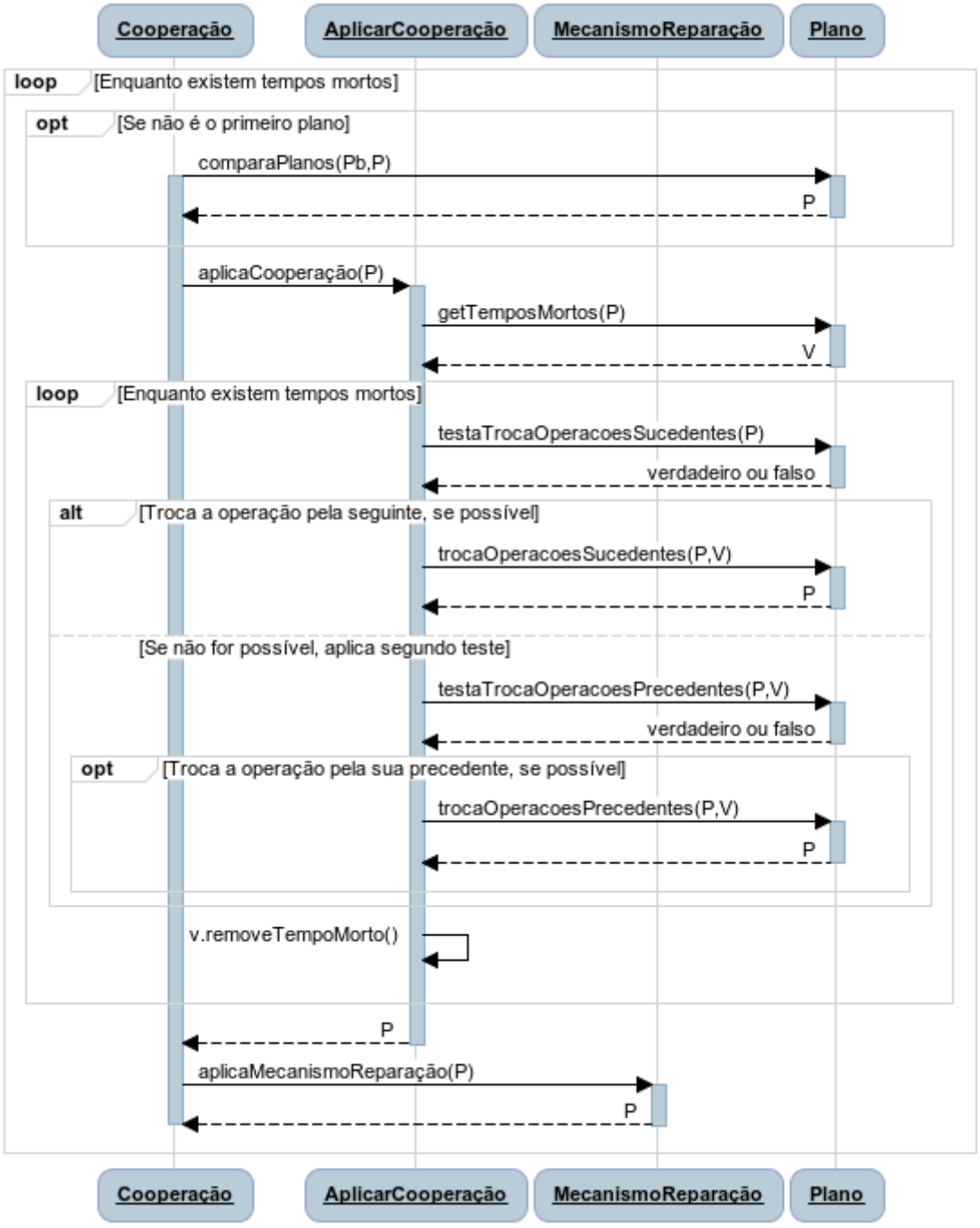

Figura 7.9 – Diagrama de sequência do Mecanismo de Cooperação (Madureira *et al.*, 2014)

É mantida uma cópia de segurança do melhor plano obtido ao longo do processo de cooperação, a qual é atualizada sempre que o mecanismo de cooperação obtém uma solução melhor do que a anterior. Este comportamento garante que, mesmo quando não melhora a solução, o Mecanismo de Cooperação tem sempre o melhor plano guardado, não permitindo a degradação da solução.

O Mecanismo de Cooperação (Figura 7.9, Algoritmo 7.1) é iniciado pelo Agente UI e começa por calcular os tempos mortos do plano atual e seguidamente calcula qual das duas possíveis ações podem ser aplicadas de modo a poder obter uma solução melhor. A primeira hipótese investiga se é possível trocar a operação atual pela seguinte operação na máquina. A segunda hipótese tenta trocar a operação pela sua precedente na tarefa e só é aplicada se não foi possível aplicar a primeira hipótese. Se não for possível aplicar nenhuma das duas hipóteses, o algoritmo recomeça mas descartando o tempo morto, passando assim para o seguinte. A cooperação termina quando não existem mais tempos mortos por tratar.

Após a troca de operações (se possível), o mecanismo devolve o plano atual, o qual necessita de ser novamente revisto/reparado para garantir que todas as restrições do problema são mantidas. O ciclo completo recomeça com o novo plano, mas antes de avançar com a cooperação é necessário comparar com o plano guardado e validar qual dos dois obteve a melhor solução global. Com este comportamento garante-se que não só a solução nunca é degradada mas que também existe um plano viável para o caso da cooperação não gerar melhorias.

O ciclo completo de cooperação (i.e. efetuar cópia de segurança, aplicar cooperação, reparar o novo plano, comparar o novo plano com a cópia) termina quando não existem mais tempos mortos para melhorar ou quando não é possível gerar um plano melhor do que o anteriormente encontrado.

7.5.2. Mecanismo de Negociação

A segunda abordagem de coordenação procura incorporar no sistema capacidades de negociação para que o plano de escalonamento seja melhorado através de redução de tempos mortos e aumento da taxa de utilização das máquinas, de forma similar ao Mecanismo de Cooperação mas desta vez com todo o trabalho a ser realizado pelos Agentes Recurso numa perspetiva competitiva.

O Mecanismo de Negociação (Madureira *et al.*, 2013c; Madureira *et al.*, 2010b; Madureira *et al.*, 2011b) é iniciado pelo Agente UI e funciona num ciclo contínuo para que todos os tempos mortos possam ser analisados. Deste modo, o mecanismo é concluído quando deixa de ser possível trocar operações e/ou quando os créditos expiram.

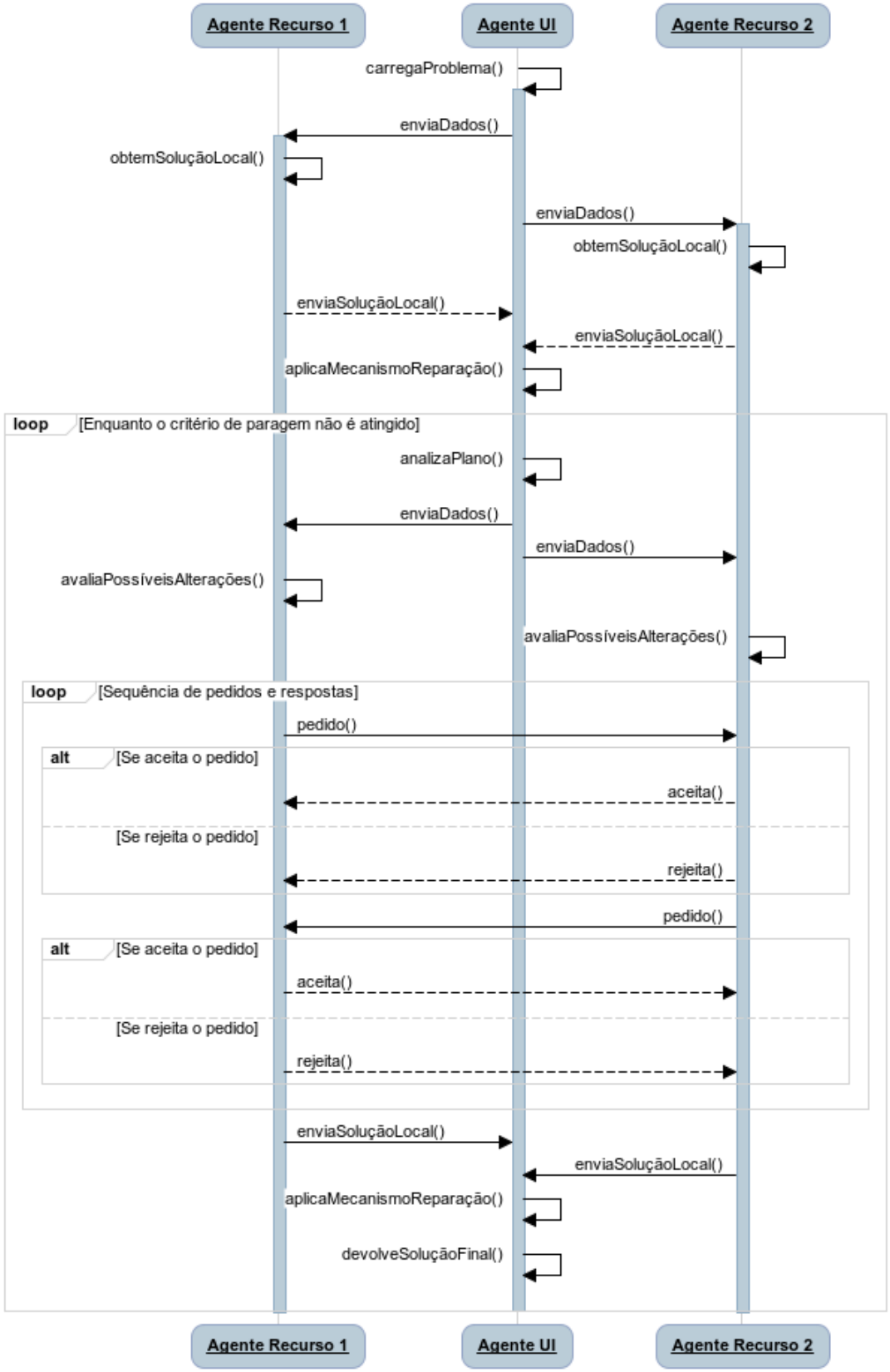

Figura 7.10 – Diagrama de sequência do Mecanismo de Negociação (Madureira *et al.*, 2013c)

Algoritmo 7.2 – Mecanismo de Negociação (Madureira *et al.*, 2013c)

```
     Entrada: ρ  // Plano resultante do Mecanismo de Reparação
     Saída: ρ // Melhor plano obtido
1    Se mecanismo_executado_pela_primeira vez() Então
2           atualiza_dados_em_cada_agente();
3           começa_comunicação_entre_agentes();
4    Fim
5    Enquanto !criterio_paragem() Faz // Executa enquanto o critério de paragem não é atingido
6           ρ ← obtem_soluções_agentes();
7           ρ ← avalia_plano(ρ); // Avalia o plano, tempos mortos e operações precedentes
8           Se !ρ.valido() Então // Se o plano não é válido
9                  atualiza_agentes_com_solução_anterior();
10                 termina_negociação();
11          Fim
12          Se melhor(ρ) Então // Se o plano atual é o melhor até ao momento
13                 atualiza_dados_em_cada_agente();
14                 recomeça_comunicação_entre_agentes();
15          Senão
16                 continuar_negociação_para_proxima_operaçao(ρ);
17          Fim
18   Fim
19   Devolver ρ;
```

O Mecanismo de Negociação (Figura 7.10, Algoritmo 7.2) procura reduzir (ou mesmo eliminar) os tempos mortos entre operações nas máquinas, aumentando assim a taxa de utilização de cada recurso, diminuindo os atrasos e o tempo de conclusão do plano. Assim sendo, o negociador tem de antecipar o processamento das operações precedentes ou recolocar outra operação para ocupar um determinado tempo morto. Este mecanismo é responsável por tratar diferentes cenários e escolher aquele que se encontra mais otimizado.

O Mecanismo de Negociação segue uma abordagem de *Contract-Net Protocol* (Smith, 1980). Neste contexto, considera-se dois tipos de agentes: um iniciador e um participante. Num determinado instante de tempo, um agente pode ser iniciador, participante, ou ambos. Neste sentido, os iniciadores são gestores e os participantes são contratados. Cada agente tem um montante de crédito que corresponde à quantidade total de tempos mortos encontrados no plano de escalonamento. Este crédito pode ser usado como moeda de troca para comprar algo aos outros agentes. Como salvaguarda, um agente que faz mudanças ao seu próprio plano recebe uma compensação que é adicionada ao crédito.

O protocolo estabelecido entre dois agentes considera que a negociação se baseia em múltiplos contactos sucessivos.

Durante este processo negocial, os agentes podem trocar as seguintes mensagens:

- Pedido(A1, A2, AçãoY) – o agente A1 pede ao agente A2 para executar uma ação (e.g. trocar a operação 1 pela operação 2);
- Aceita(A1, A2, AçãoY) – o agente A1 diz ao agente A2 que aceita o seu pedido para executar a ação;
- Rejeita(A1, A2, AçãoY) – o agente A1 comunica ao agente A2 que não aceita o pedido para executar a ação;

Neste algoritmo de negociação, cada agente tenta obter um plano de escalonamento com o maior ganho de crédito. O primeiro agente iniciador é aquele com o maior valor de tempos mortos entre operações. Após a escolha do primeiro iniciador, o processo começa com o algoritmo descrito no Algoritmo 7.2.

O Mecanismo de Negociação só é usado quando existe uma solução global resultante do Módulo de Escalonamento (baseado na integração e reparação das soluções locais dos Agentes Recurso), ou quando existe uma perturbação (chegada de novas tarefas, cancelamento de tarefas, alterações de datas de entrega, etc.) e a sua adaptação dinâmica é incorporada no plano.

7.6. Sumário

Neste capítulo foi descrito o sistema *AutoDynAgents*, que consiste num Sistema Multiagente concebido para a resolução autónoma, distribuída e cooperativa de problemas de escalonamento sujeitos a perturbações. Foi realizada a descrição de todos os tipos de agentes, incluindo os agentes de autogestão, assim como dos principais módulos presentes na sua arquitetura. Estes módulos incluem o módulo de Escalonamento, o módulo de Adaptação Dinâmica, e o módulo de Coordenação.

Capítulo 8. Módulo de Auto-Otimização: Proposta dos Mecanismos de Aprendizagem

8.1. Introdução

As Meta-heurísticas são técnicas de otimização cujo funcionamento é possível associar ao funcionamento de alguns sistemas encontrados na natureza e sistemas biológicos. Estas técnicas são bastante úteis na obtenção de boas soluções requerendo tempo computacional pouco significativo, sendo que algumas vezes podem mesmo alcançar as soluções ótimas. Mas para que estas soluções ótimas ou quase-ótimas possam ser atingidas, é necessária a correta definição dos parâmetros, tarefa que requer algum conhecimento pericial da Meta-heurística a usar e do problema a tratar. Muitas vezes, para afinar os parâmetros, é usado o método de tentativa-erro, e a dificuldade aumenta quando existe a possibilidade de se poder usar mais do que uma técnica. Neste caso, é necessário selecionar a Meta-heurística *a priori* e só depois proceder à definição dos parâmetros.

Um dos objetivos do sistema *AutoDynAgents* é ser capaz de adotar e fornecer a capacidade de autoparametrização para as Meta-heurísticas, de acordo com a instância do problema de escalonamento a ser resolvida. O sistema deve ser capaz de escolher uma Meta-heurística a usar e definir os seus parâmetros de entrada, de acordo com as características da situação atual (e.g. dimensão e complexidade da instância do problema). Além disso, permite comutar duma técnica para outra, dependendo da instância a tratar e da experiência passada acumulada. A autoparametrização deverá ser efetuada através de aprendizagem baseada na experiência.

Para resolver o problema de autoparametrização, foi proposto e desenvolvido um agente incorporando ideias da Computação Autónoma, designado de Auto-Otimização (subsecção 5.2.2). Este agente é capaz de monitorizar o sistema e configurar os parâmetros das diferentes Meta-heurísticas, considerando as características de cada problema que surge no sistema. Assim, o agente deve saber como parametrizar cada Meta-heurística, e, uma vez que é impossível prever todos os problemas a tratar, deve ser capaz de aprender com a experiência com base em problemas anteriores, à semelhança dos humanos.

Neste capítulo pretende-se dar uma contribuição para a resolução do problema de autoparametrização de Meta-heurísticas na resolução do problema de escalonamento de

tarefas de produção. Neste sentido, é proposto um mecanismo baseado em *Racing* e outro em Raciocínio baseado em Casos, para implementar a capacidade de autoparametrização. É ainda proposto um mecanismo híbrido no sentido de tirar partido das potencialidades das duas técnicas de aprendizagem. O *Racing* permite que seja efetuado um estudo entre várias combinações de parâmetros, para que seja possível determinar a melhor combinação na aplicação de uma determinada Meta-heurística. O Raciocínio baseado em Casos dá ao sistema a capacidade de evoluir e aprender com a experiência, uma vez que se baseia em casos passados e retém toda a informação da experiência para uso futuro.

8.2. Descrição geral

Nesta secção, são descritas as duas diferentes estratégias de aprendizagem que foram consideradas, explica-se a arquitetura global do módulo de Auto-Otimização, e descreve-se a base de dados (e casos) utilizada para dar suporte aos mecanismos de aprendizagem *Racing* e Raciocínio baseado em Casos.

8.2.1. Estratégias de aprendizagem

Numa fase preliminar deste trabalho de doutoramento, foi realizado um esforço na validação do desempenho de duas abordagens de Aprendizagem Cooperativa Multiagente (Panait e Luke, 2005). A primeira, Aprendizagem para a Equipa, considera apenas um aprendiz que tenciona aprender por toda a comunidade de agentes. A segunda estratégia, Aprendizagem Concorrente, utiliza múltiplos processos de aprendizagem, tipicamente um por cada agente da comunidade.

Neste sentido, foi realizado um estudo computacional (secção 9.2), tendo-se evidenciado vantagem na abordagem de Aprendizagem para a Equipa. Tal poderá justificar-se pelo facto do agente aprendiz possuir uma visão global de todo o sistema, tentando otimizar o plano de escalonamento como um todo. Na abordagem de Aprendizagem Concorrente, os vários agentes aprendizes são egoístas e competitivos entre si, de modo a tentar melhorar o seu plano local, não se preocupando com o plano global.

Por este motivo, decidiu-se realizar a proposta e desenvolvimento dos mecanismos de aprendizagem no âmbito de uma arquitetura baseada em Aprendizagem para a Equipa. No entanto, para que o leitor possa compreender as diferenças existentes, são aqui apresentadas as duas abordagens.

8.2.1.1. Aprendizagem para a Equipa

Numa abordagem de Aprendizagem para a Equipa, existe um único agente aprendiz, com o objetivo de aprender e descobrir um conjunto de comportamentos para uma equipa de agentes. No entanto, o agente aprendiz preocupa-se em aprender e melhorar o desempenho global da equipa, e não consigo próprio.

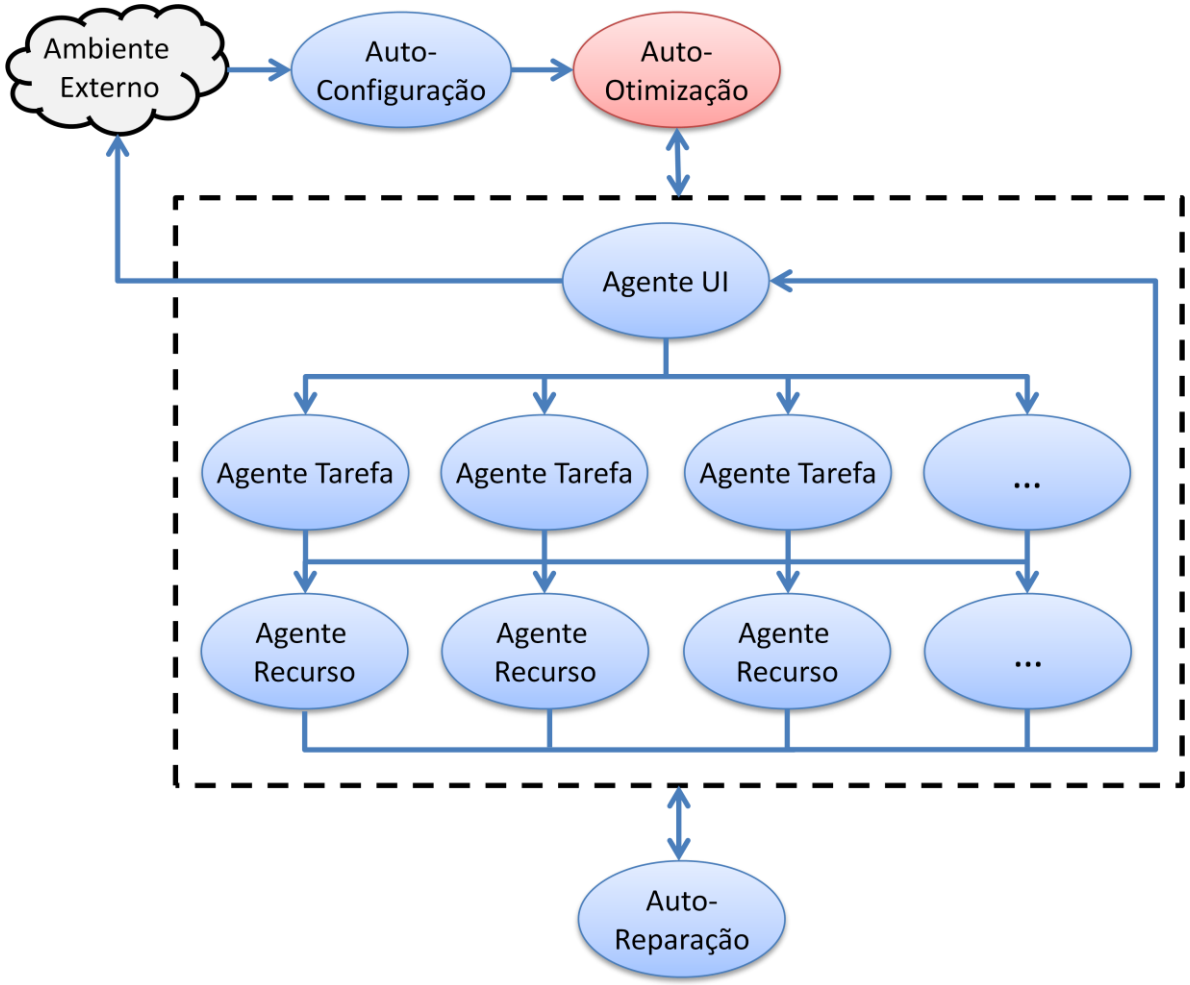

Figura 8.1 – Ilustração da abordagem de Aprendizagem para a Equipa

Na Figura 8.1 encontra-se ilustrada a abordagem de Aprendizagem para a Equipa considerada. Por questões de simplicidade, apenas se mostram os agentes envolvidos no processo de aprendizagem. O agente aprendiz (Auto-Otimização) comunica com o AgenteUI, de modo a ter uma visão global do sistema. Inicialmente, tem acesso a todos os dados da instância do problema de escalonamento, e.g. número de tarefas, número de máquinas/recursos, tipo de problema a resolver, etc. Nesta fase, o agente de Auto-Otimização sugere uma Meta-heurística e respetiva parametrização para aplicar na resolução da instância do problema. No final da obtenção do plano de escalonamento, o

agente aprendiz tem acesso ao plano de escalonamento e pode analisar o desempenho global para, desse modo, aprender.

8.2.1.2. Aprendizagem Concorrente

Em oposição à abordagem de Aprendizagem para a Equipa, considerou-se uma abordagem de Aprendizagem Concorrente, que emprega vários aprendizes dentro da comunidade de agentes. Considera-se que uma abordagem de Aprendizagem Concorrente é muito mais competitiva do que uma abordagem de Aprendizagem para a Equipa.

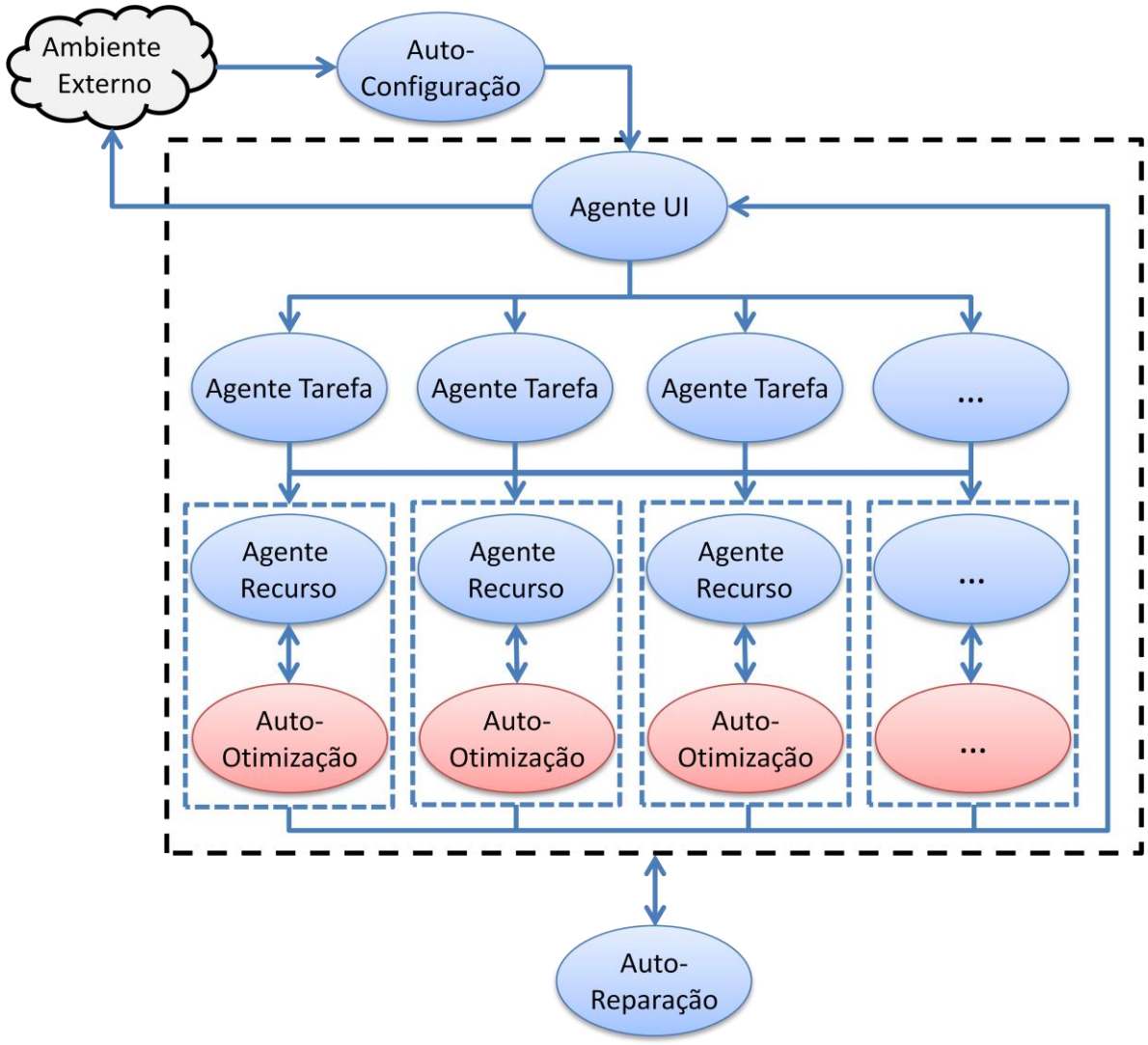

Figura 8.2 – Ilustração da abordagem de Aprendizagem Concorrente (Pereira *et al.*, 2012; Pereira *et al.*, 2013b)

Na abordagem de Aprendizagem Concorrente (Figura 8.2), o processo de obtenção de planos de escalonamento é idêntico à abordagem de Aprendizagem para a Equipa, com

a diferença que não existe aprendizagem a um nível superior, mas sim ao nível dos agentes recurso. Cada Agente Recurso possui um módulo de Auto-Otimização e é responsável por aprender e melhorar o problema de máquina única respetivo. Assim, cada Agente Recurso tem uma visão local e apenas consegue melhorar o seu próprio desempenho, não tendo perceção sobre o desempenho global do sistema.

8.2.2. Arquitetura global

Na Figura 8.3 encontra-se ilustrada genericamente a arquitetura do agente de Auto-Otimização. Foram definidas três formas de funcionamento:

1. *Racing* (a azul) - o agente de Auto-Otimização efetua o estudo de *Racing*, tendo como resultado os melhores parâmetros para cada Meta-heurística;
2. Raciocínio baseado em Casos (a verde) - o agente recebe uma parametrização inicial para as técnicas e, a partir dela, aprende através da experiência;
3. *Racing* + Raciocínio baseado em Casos (a vermelho) - o estudo de *Racing* é utilizado para especificar uma parametrização inicial ao Raciocínio baseado em Casos e depois o sistema evolui naturalmente através da experiência.

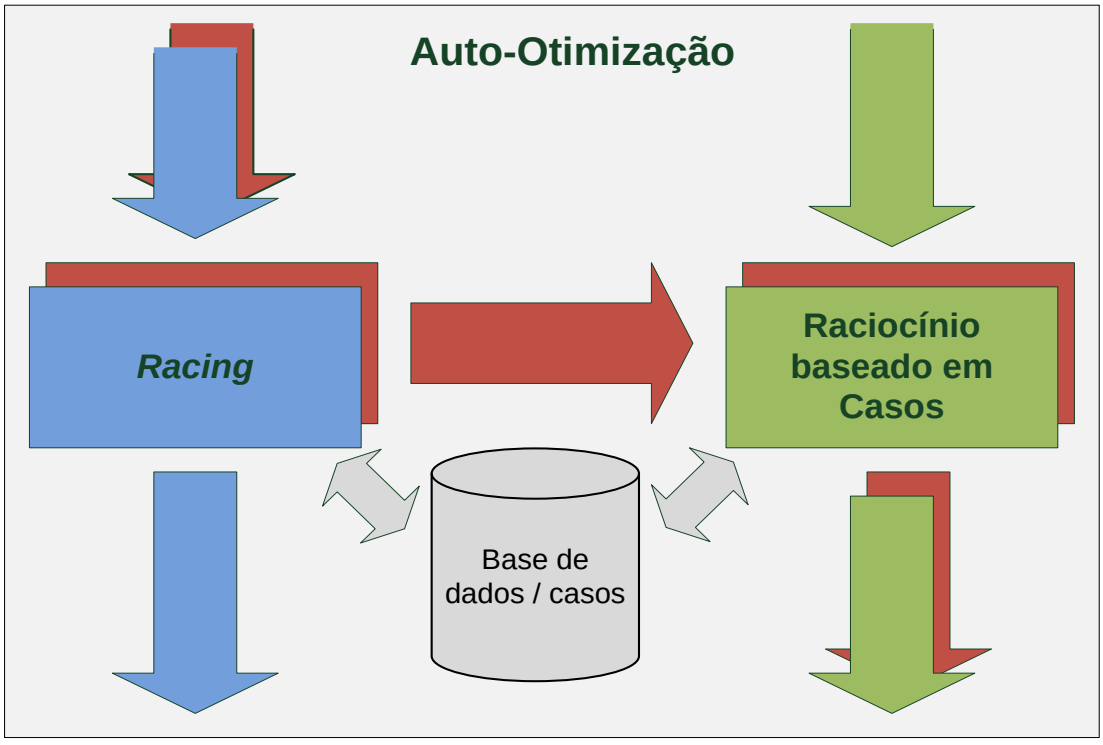

Figura 8.3 – Arquitetura global do agente de Auto-Otimização

A decisão sobre qual o comportamento a considerar pelo agente de Auto-Otimização cabe ao utilizador/perito, sendo que o objetivo final é a utilização de *Racing* + Raciocínio

baseado em Casos. No entanto, é importante salientar que, o estudo de *Racing* é efetuado apenas no início, de forma a poder a inicializar a base de casos para o Raciocínio baseado em Casos, sendo estas parametrizações resultantes de um estudo realizado de forma autónoma, como descrito na secção 8.3.

8.2.3. Base de dados/casos

A base de dados utilizada está preparada para guardar toda a informação dos estudos de *Racing* e todos os casos do módulo de Raciocínio baseado em Casos. É composta por doze tabelas e encontra-se ilustrada na Figura 8.4.

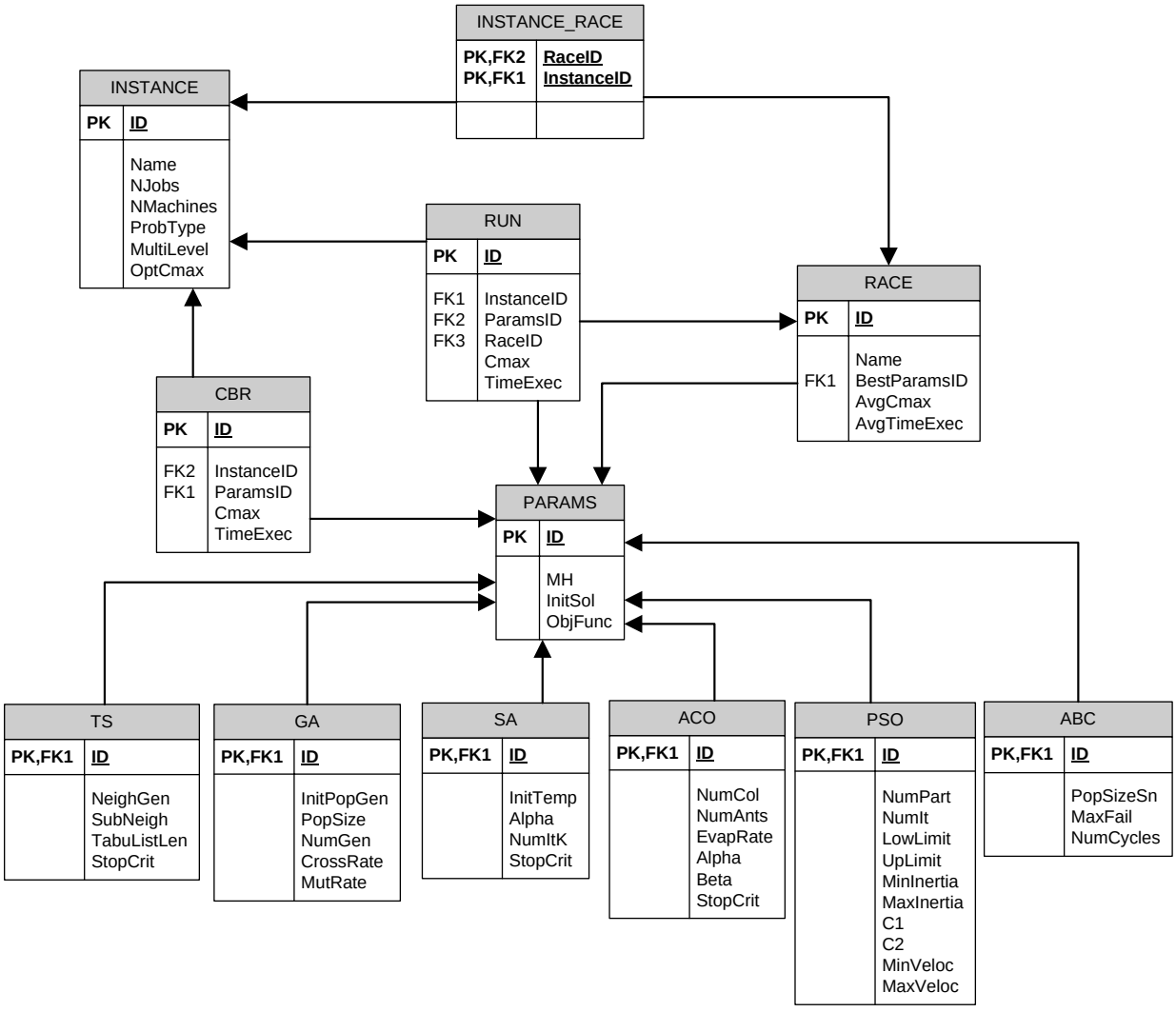

Figura 8.4 – Base de dados/casos

A tabela *Instance* (Tabela 8.1) guarda todos os dados relativos à instância do problema a ser tratado. Considera-se que uma instância de um problema de escalonamento é caracterizada pelo número de tarefas, pelo número de máquinas, pelo tipo de problema, pelo facto de existirem relações multinível[8] e também pelo seu tempo de conclusão ótimo[9].

Tabela 8.1 – Tabela *Instance*

Campo	Tipo de dados	Definição
ID	UUID[10]	Identificador único da instância
Name	String	Nome do ficheiro da instância de problema
NJobs	Integer	Número de tarefas da instância
NMachines	Integer	Número de máquinas da instância
ProbType	String	Tipo de problema (*Single-Machine*, *Open-Shop*, *Flow-Shop*, ou *Job-Shop*)
MultiLevel	Boolean	Indica se a instância corresponde ou não a um problema multinível, ou seja se tem operações com mais do que um precedente
OptCmax	Integer	Valor ótimo de tempo de conclusão (*Cmax*) conhecido. Se não for conhecido então não é considerado.

Tabela 8.2 – Tabela *Race*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único da *Race*
Name	String	Nome da *Race*, e.g. "Ts 10 cmax" que significa que a *Race* se destina a avaliar combinações de parâmetros para Pesquisa Tabu em instâncias de 10 tarefas, na minimização de C_{max}
BestParamsID	UUID	Referência para a melhor combinação de parâmetros obtida no final da *Race*
AvgCmax	Double	Tempo de conclusão (C_{max}) médio obtido pela melhor combinação de parâmetros
AvgTimeExec	Double	Tempo de execução médio obtido pela melhor combinação de parâmetros

A tabela *Race* (Tabela 8.2) diz respeito unicamente ao módulo de *Racing* e é responsável por guardar os dados respeitantes à melhor combinação de parâmetros encontrada pelo estudo de *Racing* da Meta-heurística.

[8] Quando uma operação pode ser precedida e/ou sucedida por mais do que uma operação

[9] Para as instâncias de *benchmark* usadas neste trabalho, a literatura disponibiliza os valores das melhores soluções conhecidas até ao momento

[10] Identificador universal único (*Universally Unique Identifier*) usado para fornecer uma referência única para o registo

Tabela 8.3 – Tabela *Run*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único da *Run*
InstanceID	UUID	Referência para a instância a que corresponde esta *Run*
ParamsID	UUID	Referência para a combinação de parâmetros a que corresponde esta *Run*
RaceID	UUID	Referência para a *Race* a qual corresponde esta *Run*
Cmax	Integer	Tempo de conclusão (C_{max}) obtido
TimeExec	Double	Tempo de execução obtido

A tabela *Run* (Tabela 8.3) cria um registo por cada execução do estudo de *Racing*, fazendo referência à instância resolvida, à combinação de parâmetros usada, e aos resultados obtidos.

A tabela *Cbr* (Tabela 8.4) sistematiza a informação do módulo de Raciocínio baseado em Casos e cria um registo para cada caso na base de dados/caso. Cada caso é caracterizado pelos dados da instância tratada, pelos parâmetros da Meta-heurística usada e pelos resultados obtidos (tempo de conclusão e tempo de execução).

Tabela 8.4 – Tabela *Cbr*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único do caso
InstanceID	UUID	Referência para a instância a que corresponde este caso
ParamsID	UUID	Referência para a combinação de parâmetros usada por este caso
Cmax	Integer	Tempo de conclusão (C_{max}) obtido
TimeExec	Double	Tempo de execução obtido

Tabela 8.5 – Tabela *Params*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único da combinação de parâmetros
MH	String	Nome da tabela da Meta-heurística a utilizar (TS, GA, SA, ACO, PSO, ou ABC)
InitSol	String	Heurística construtiva usada para construir a solução inicial (*SeqNivel*, EDD, SPT, REDD, ou RND)
ObjFunc	String	Função objetivo usada para minimização do problema (C_{max}, WT, ou L_{max})

A tabela *Params* (Tabela 8.5) é uma tabela genérica, usada por ambos os módulos, para fazer a ligação aos parâmetros da Meta-heurística usada. Assim, nesta tabela é guardado o nome da técnica usada bem como a heurística de construção da solução inicial e a função objetivo a ser otimizada.

As restantes tabelas sistematizam as parametrizações/soluções para cada combinação de parâmetros, ou seja, os parâmetros de cada Meta-heurística. Assim, existe uma tabela por cada Meta-heurística usada no sistema, i.e. Pesquisa Tabu (*Ts*), Algoritmos Genéticos (*Ga*), *Simulated Annealing* (*Sa*), Otimização por Colónia de Formigas (*Aco*), *Particle Swarm Optimization* (*Pso*), e Colónia de Abelhas Artificiais (*Abc*), sendo estas descritas na Tabela 8.6, Tabela 8.7, Tabela 8.8, Tabela 8.9, Tabela 8.10, e Tabela 8.11.

Tabela 8.6 – Tabela *Ts*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único da Meta-heurística, que tem relação de 1-para-1 com o caso
NeighGen	Double	Percentagem de geração de vizinhança, vulgo afastamento máximo
SubNeigh	Double	Percentagem de subvizinhança
TabuListLen	Integer	Tamanho da lista tabu
StopCrit	Integer	Critério de paragem, i.e. número de iterações

Tabela 8.7 – Tabela *Ga*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único da Meta-heurística, que tem relação de 1-para-1 com o caso
InitPopGen	Double	Percentagem de geração da população inicial, vulgo afastamento máximo
PopSize	Double	Percentagem do tamanho da população inicial
NumGen	Integer	Número de gerações
CrossRate	Double	Taxa de cruzamento
MutRate	Double	Taxa de mutação

Tabela 8.8 – Tabela *Sa*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único da Meta-heurística, que tem relação de 1-para-1 com o caso
InitTemp	Double	Temperatura inicial
Alpha	Double	Fator de redução da temperatura (arrefecimento)
NumItK	Integer	Número de iterações à mesma temperatura
StopCrit	Integer	Critério de paragem, i.e. número de iterações

Tabela 8.9 – Tabela *Aco*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único da Meta-heurística, que tem relação de 1-para-1 com o caso
NumCol	Integer	Número de colónias
NumAnts	Integer	Número de formigas por colónia
EvapRate	Double	Taxa de evaporação da feromona
Alpha	Double	Importância do valor heurístico
Beta	Double	Importância da feromona
StopCrit	Integer	Critério de paragem, i.e. número de iterações

Tabela 8.10 – Tabela *Pso*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único da Meta-heurística, que tem relação de 1-para-1 com o caso
NumPart	Integer	Número de partículas
NumIt	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 8.11 – Tabela *Abc*

Campo	Tipo de dados	Definição
ID	UUID	Identificador único da Meta-heurística, que tem relação de 1-para-1 com o caso
PopSizeSn	Integer	Tamanho da população
MaxFail	Integer	Número máximo de falhas
NumCycles	Integer	Número de ciclos iterativos

8.3. Módulo de *Racing*

O módulo de *Racing*, descrito nesta secção, surge de uma adaptação do método F-Race (subsecção 6.4.1.2). A impossibilidade de efetuar uma implementação direta do algoritmo de eliminação de candidatos, por não existir um número suficiente de instâncias de escalonamento, levou à necessidade de adaptar e implementar uma solução onde o funcionamento geral fosse similar ao algoritmo genérico de *Racing*, descrito no Algoritmo 6.2.

8.3.1. Funcionamento geral

O objetivo do módulo de *Racing* passa por realizar um estudo de combinações de parâmetros de várias Meta-heurísticas na otimização de várias funções objetivo. O Algoritmo 8.1 descreve em pseudocódigo os passos para a realização deste estudo. Assim, os parâmetros de entrada referem-se à lista de funções objetivo a otimizar e a lista de Meta-heurísticas a validar.

Percorrendo as funções objetivo, o primeiro passo passa por obter uma lista de parâmetros candidatos de cada Meta-heurística para efetuar "*corridas*" entre eles. Estas "*corridas*" são realizadas num determinado número de instâncias. Para cada instância, cada combinação de parâmetros é testada no sistema, e os dados são guardados na base de dados. No final de cada instância, os candidatos que obtiveram piores resultados são eliminados. No final da "*corrida*", o melhor candidato é aquele que foi capaz de sobreviver ao longo das várias instâncias.

O algoritmo mais relevante de todo este processo é o método *eliminarCandidatos()*, descrito no Algoritmo 8.2, uma vez que é este que decide quais os candidatos que vão ser eliminados da "*corrida"*. Este algoritmo recebe como parâmetros de entrada a *race* atual e a lista de parâmetros candidatos, devolvendo a lista de parâmetros sobreviventes. A primeira verificação a realizar é verificar se a lista de parâmetros candidatos tem mais do que uma combinação, caso contrário estamos perante o melhor caso. De seguida, é necessário verificar qual o teste estatístico a ser usado. Se a lista de candidatos tiver mais do que dois elementos, aplica-se o teste de Friedman. Caso contrário, utiliza-se o teste de Wilcoxon.

Algoritmo 8.1 – Algoritmo de *Racing*

```
    // Lista de funções objetivo, lista de Meta-heurísticas
    Entrada: listaObjFunc, listaMH
    Saída:
1   inicializar(listaInstancias); // Array para guardar as instâncias
2   inicializar(listaParams); // Array para guardar os parâmetros candidatos
3   Para cada objFunc ∈ listaObjFunc Faz // Percorrer todas as funções objetivo
4           // Obter as instâncias do problema
5           listaInstancias ← getInstancias();
6           Para cada mh ∈ listaMH Faz // Percorrer todas as Meta-heurísticas
7                   // Obter a lista de parâmetros candidatos
8                   listaParams ← getParams(objFunc, mh);
9                   // Criar a Race para avaliar a lista de parâmetros candidatos
10                  race ← criarRace(listaParams);
11                  // Percorrer todas as instâncias
12                  Para cada instancia ∈ listaInstancias Faz
13                          // Percorrer todas as combinações de parâmetros candidatos
14                          Para cada params ∈ listaParams Faz
15                                  //Criar Run para avaliar a combinação de parâmetros
16                                  run ← criarRun(race,instancia,params);
17                                  executarRun(run);
18                          Fim
19                          // Eliminar os candidatos piores
20                          listaParams ← eliminarCandidatos(race,listaParams);
21                  Fim
22                  // Obter a melhor combinação de parâmetros sobrevivente
23                  melhorCandidato ← race.getBestCandidate();
24                  // Calcular e guardar as médias dos valores do melhor candidato
25                  race.calcularGuardarMedias(melhorCandidato);
26          Fim
27  Fim
28  Termina
```

Algoritmo 8.2 – Método *eliminarCandidatos()*

```
    Entrada: race, listaParams // Race atual e lista de parâmetros candidatos
    Saída: listaParams // Lista de parâmetros sobreviventes
1   Se tamanho(listaParams) > 1 Então
2           listaInstancias ← race.getInstancias(); // Obter lista de instâncias da Race
3           Se tamanho(listaParams) > 2 Então
4                   // Só aplica o teste de Friedman se houver mais do que dois candidatos
5                   listaParams ← aplicaTesteFriedman(listaParams,listaInstancias);
6           Senão
7                   // Caso contrário aplica o teste de Wilcoxon
8                   listaParams ← aplicaTesteWilcoxon(listaParams,listaInstancias);
9           Fim
10  Fim
11  Devolver listaParams
```

O teste de Friedman[11] (Algoritmo 8.3) necessita que existam no mínimo dois blocos de resultados, i.e. é necessário que todos os candidatos tenham sido executados em pelo menos duas instâncias. Tal implica que, no final da primeira instância todos os candidatos sobrevivam a este método. Caso seja possível aplicar o teste de Friedman, primeiro é necessário obter os rankings ordenados dos parâmetros candidatos, sendo de seguida calculado o somatório de cada um para cada candidato. O número de sobreviventes serão os *n* melhores, calculados através da equação (8.1).

Algoritmo 8.3 – Teste de Friedman

```
    // Lista de parâmetros candidatos e lista de instâncias
    Entrada: listaParams, listaInstancias
    Saída: listaParams // Lista de parâmetros sobreviventes
1   inicializar(ranks); // Matriz de rankings dos parâmetros candidatos por instância
2   inicializar(somaRanks); // Array da soma dos rankings dos parâmetros candidatos
3   inst ← tamanho(listaInstancias); // Número de instâncias
4   cand ← tamanho(listaParams); // Número de candidatos
5   // Só aplica o teste se houver resultados para mais do que uma instância
6   Se i > 1 Então
7           // Obtém os rankings ordenados dos parâmetros segundo o teste de Friedman
8           ranks ← ordenarRanksFriedman(listaInstancias, listaParams);
9           // Calcula o somatório dos rankings para cada candidato
10          somaRanks ← somarRanks(ranks);
11          // Calcula o número de parâmetros sobreviventes com base na equação (8.1)
12          sobrev ← calcularNumSobrev(inst, cand);
13          // Obtém a lista de parâmetros sobreviventes
14          listaParams ← obterSobreviventes(sobrev, somaRanks);
15  Fim
16  Devolver listaParams
```

[11] Implementado com recurso à biblioteca JSC (*Java Statistical Classes*): http://www.jsc.nildram.co.uk/

O número de sobreviventes (equação (8.1)) em cada passo é dependente do número de instâncias já executadas e do número de candidatos. Além disso, é calculado em função do inverso do logaritmo de base 2, o que permite obter um comportamento similar ao *F-Race*, tal como ilustrado na Figura 8.5. Com esta função, é possível, mesmo para um número reduzido de instâncias, obter uma rápida convergência para o melhor candidato, ao contrário do que acontece com o *F-Race* original.

$$sobrev = round\left(\frac{1}{\log_2(inst+1)} \times cand\right) \quad (8.1)$$

onde *inst* corresponde ao número de instâncias e *cand* corresponde ao número de candidatos.

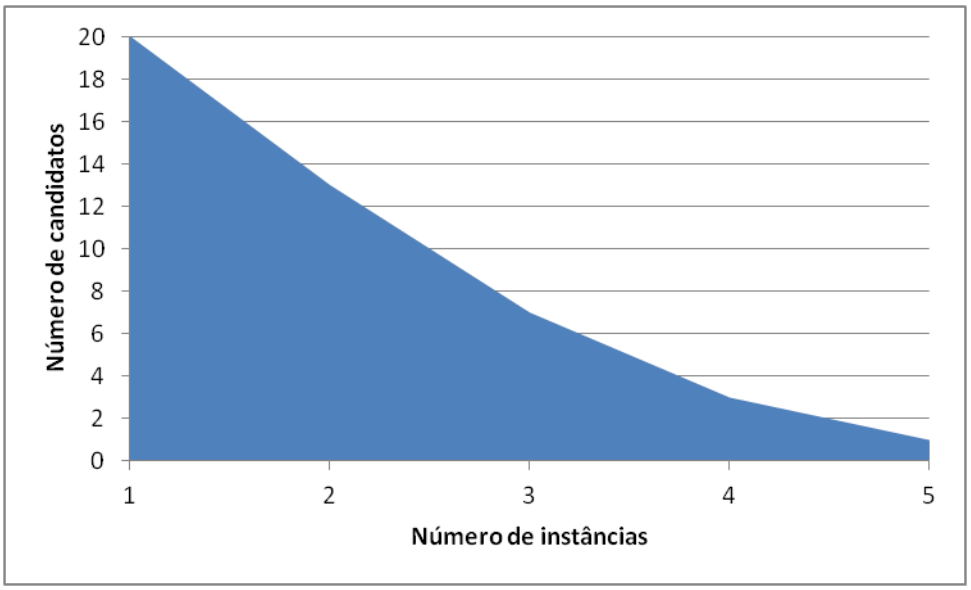

Figura 8.5 – Representação gráfica da evolução do número de candidatos com base na função $1/\log_2$ (20 candidatos em 5 instâncias)

Da Figura 8.5 é possível analisar a evolução do número de candidatos em função do número de instâncias, com base na função $1/\log_2$.

Quando existem apenas dois candidatos, utiliza-se o teste de Wilcoxon[12] em vez do teste de Friedman. O teste de Wilcoxon (Algoritmo 8.4) compara dois candidatos de forma ligeiramente diferente. Primeiro, é necessário obter um *array* com a diferença de valores de execução nas várias instâncias entre o primeiro e o segundo candidato. Seguidamente, é realizado um *ranking* do valor absoluto destas diferenças, para depois ser possível calcular um valor *w* resultante do somatório ponderado entre os *ranks* e os sinais da diferença de valores (equação (8.2)).

[12] Implementado com recurso à biblioteca *Commons Math*: http://commons.apache.org/proper/commons-math/

$$w = \sum_{j=0}^{inst} ranks[j] \times signs[j] \quad (8.2)$$

onde *inst* corresponde ao número de instâncias.

Se o valor *w* for inferior a zero, significa que o primeiro candidato é melhor, caso contrário, o melhor será o segundo candidato. Em qualquer dos casos, o pior candidato é eliminado. Este procedimento será descrito com um exemplo ilustrativo.

Algoritmo 8.4 – Teste de Wilcoxon

```
    // Lista de parâmetros candidatos e lista de instâncias
    Entrada: listaParams, listaInstancias
    Saída: listaParams // Lista de parâmetros sobreviventes
 1  inst ← tamanho(listaInstancias); // Número de instâncias
 2  // Array com os valores dos resultados por instância do primeiro candidato
 3  a ← obterResultados(listaInstancias, listaParams.get(0));
 4  // Array com os valores dos resultados por instância do segundo candidato
 5  b ← obterResultados(listaInstancias, listaParams.get(1));
 6  // Array para guardar a diferença de valores entre o primeiro e o segundo candidato
 7  dif ← calcularDiferenca(a, b);
 8  absDif ← abs(dif); // Array para guardar o valor absoluto da diferença de valores
 9  ranks ← calcularRanks(absDif); // Array para guardar os rankings das diferenças
10  signs ← sinal(dif); // Array para guardar o sinal da diferença de valores
11  // Valor usado para validar o teste de Wilcoxon, calculado pela equação (8.2)
12  w ← calcularW(ranks, signs, inst);
13  Se w < 0 Então
14          listaParams.remove(1); // O primeiro candidato é melhor, elimina o segundo
15  Senão
16          listaParams.remove(0); // O segundo candidato é melhor, elimina o primeiro
17  Fim
18  Devolver listaParams
```

8.3.2. Exemplo ilustrativo

Com este exemplo, pretende-se ilustrar a execução de um algoritmo de *Racing* na obtenção da melhor combinação de parâmetros numa corrida com quatro instâncias. Para ilustrar o funcionamento do módulo de *Racing*, considerem-se os valores para os parâmetros da técnica de otimização Pesquisa Tabu definidos na Tabela 8.12.

Tabela 8.12 – Possíveis valores para os parâmetros da Pesquisa Tabu

	NeighGen	SubNeigh	TabuListLen	StopCrit
TS	15%	100%	1 3	50 75 100

Tabela 8.13 – Combinações de parâmetros da Pesquisa Tabu

	NeighGen	SubNeigh	TabuListLen	StopCrit
P1	15%	100%	1	50
P2	15%	100%	1	75
P3	15%	100%	1	100
P4	15%	100%	3	50
P5	15%	100%	3	75
P6	15%	100%	3	100

Com base nos possíveis valores para os quatro parâmetros da Pesquisa Tabu, é possível chegar a seis diferentes combinações, descritas na Tabela 8.13.

Neste exemplo, pretende-se então testar estas seis combinações de parâmetros ao longo de uma corrida com quatro instâncias. É importante salientar que todas as combinações de parâmetros são executadas nas duas primeiras instâncias. Apenas após a segunda instância é que é executado o algoritmo para eliminar combinações candidatas, uma vez que são necessários pelo menos dois blocos de resultados.

Tabela 8.14 – Passo 1 (Friedman)

	Resultado *Cmax*		*Ranks*		*somaRanks*
	Instância1	**Instância2**	**Instância1**	**Instância2**	
P1	791	948	2	5	7
P2	883	951	5	6	11
P3	842	932	3	4	7
P4	858	832	4	1	5
P5	894	893	6	2	8
P6	740	905	1	3	4

Na Tabela 8.14 está descrito o primeiro passo deste algoritmo. Como existem mais do que dois candidatos, é aplicado o teste de Friedman. Neste teste, é obtido um *ranking* dos candidatos para cada instância, e só depois é feita a soma dos *rankings* respetivos para cada combinação candidata, sendo a qualidade do candidato inversamente proporcional a esta soma. Neste passo, os candidatos são ordenados por ordem decrescente de *rankings* da seguinte forma: P6, P4, P1, P3, P5, P2.

Como existem seis candidatos e estamos perante a segunda instância, o número de candidatos que passam para o próximo passo é calculado pela equação (8.1):

$$sobrev = round\left(\frac{1}{\log_2(2+1)} \times 6\right) = 4\ (3{,}786)$$

Assim, os quatro melhores candidatos passam à próxima fase: P1, P3, P4, e P6. Estas quatro combinações de parâmetros são executadas numa terceira instância e é repetido o procedimento de Friedman, pois existem mais do que dois candidatos. Na Tabela 8.15 apresentam-se os dados do segundo passo.

Tabela 8.15 – Passo 2 (Friedman)

	Resultado Cmax			Ranks			somaRanks
	Instância1	**Instância2**	**Instância3**	**Instância1**	**Instância2**	**Instância3**	
P1	791	948	958	2	4	1	7
P2	842	932	961	3	3	2	8
P4	858	832	1091	4	1	4	9
P6	740	905	1032	1	2	3	6

Neste passo, os candidatos são ordenados por ordem decrescente de *rankings* da seguinte forma: P6, P1, P2, P4. Existindo quatro candidatos e estando perante a terceira instância, são dois os candidatos que passam para a próxima fase (equação (8.1)). Assim, os dois melhores candidatos são o P1 e o P6.

$$sobrev = round\left(\frac{1}{\log_2(3+1)} \times 4\right) = 2$$

Na Tabela 8.16 encontram-se os resultados obtidos pela execução destes dois candidatos nas quatro instâncias.

Tabela 8.16 – Passo 3 (Wilcoxon)

	Resultado Cmax			
	Instância1	**Instância2**	**Instância3**	**Instância4**
P1	791	948	958	873
P6	740	905	1032	927

Estando perante apenas dois candidatos, no terceiro e último passo é aplicado o teste de Wilcoxon. Os candidatos são comparados da seguinte forma (Tabela 8.17): primeiro

calcula-se a diferença de resultados entre eles, em todas as instâncias; depois é obtida uma ordem (*ranking*) do valor absoluto desta diferença; com base nos *rankings* e no sinal (positivo ou negativo) da diferença, é calculado um valor (*w*) que é usado para decidir qual é o melhor entre os dois candidatos (equação (8.2)).

$$\begin{aligned} w = \quad & 2 \times 1 \; + \\ & 1 \times 1 \; + \\ & 4 \times -1 \; + \\ & 3 \times -1 \\ w = \quad & -4 \end{aligned}$$

Tabela 8.17 – Teste de Wilcoxon

dif	51	43	-74	-54
absDif	51	43	74	54
ranks	2	1	4	3
signs	1	1	-1	-1

Sendo o valor de *w* inferior a 0, o primeiro candidato é considerado melhor e como consequência o segundo candidato é descartado. Termina assim o algoritmo de *Racing*, com a conclusão que o melhor candidato deste exemplo é o P1, cujos parâmetros estão apresentados na Tabela 8.18.

Tabela 8.18 – Melhor combinação de parâmetros

	NeighGen	**SubNeigh**	**TabuListLen**	**StopCrit**
P1	15%	100%	1	50

8.4. Módulo de Raciocínio baseado em Casos

A abordagem de Raciocínio baseado em Casos proposta, consiste em recuperar o caso mais similar com o novo problema, independentemente da Meta-heurística a usar. Assim, é retornado o caso contendo a Meta-heurística e os respetivos parâmetros a usar. Esta abordagem revela-se adequada ao problema em questão uma vez que é importante que o sistema possa decidir qual a Meta-heurística a usar e quais os respetivos parâmetros de configuração

8.4.1. Funcionamento geral

O módulo de Raciocínio baseado em Casos consiste num ciclo similar ao descrito na subsecção 6.4.2.2. Este ciclo é composto por quatro processos principais, “os quatro R’s”, i.e. Recuperação, Reutilização, Revisão e Retenção (ver Algoritmo 8.5). De notar que, na fase de Revisão, existe a comunicação com o Sistema Multiagente, de modo a ser possível executar o novo caso e retirar resultados relativamente aos tempos de conclusão e execução.

Algoritmo 8.5 – Raciocínio baseado em Casos

```
    Entrada: novoCaso
    Saída:
1   listaCasos ← recuperar(novoCaso); // Fase de Recuperação
2   melhorCaso ← reutilizar(listaCasos); // Fase de Reutilização
3   metaHeuristica ← devolverSolucao(melhorCaso);
4   simMelhorCaso ← obterSimilaridade(melhorCaso);
5   metaHeuristicaRevista ← rever(metaHeuistica, simMelhorCaso); // Fase de Revisão
6   // Executar o novo caso no Sistema Multiagente, com recurso à Meta-heurística sugerida
7   resultados ←executarCasoSMA(novoCaso, metaHeuristicaRevista);
8   reter(novoCaso, metaHeuristicaRevista, resultados); // Fase de Retenção
9   Termina
```

Sempre que surge uma nova instância para resolver, é criado um novo caso no módulo de Raciocínio baseado em Casos. Este novo caso é resolvido através da recuperação de um ou mais casos anteriores similares. Depois de se obter uma lista com os casos mais similares, é reutilizado o caso mais similar de entre todos, sendo este sugerido como possível solução. Na fase seguinte, e antes de se proceder à comunicação com o Sistema Multiagente, é realizada a revisão da solução sugerida, através da respetiva adaptação e refinamento. Por último, o caso testado é aceite como uma solução final, que por sua vez é retida na base de casos como um novo caso aprendido. É importante salientar que, caso não existam casos anteriores similares, o módulo utiliza os parâmetros pré-definidos pelo utilizador/perito, e, em caso de sucesso, retém o caso na base de casos com essa parametrização.

A fase de Recuperação tem como objetivo pesquisar a base de casos, encontrar os casos passados mais similares com o novo caso, e recuperá-los para análise, de modo a ser posteriormente selecionado um dos melhores casos e reutilizá-lo na próxima fase.

Esta fase encontra-se descrita em pseudocódigo no Algoritmo 8.6, o qual recebe o novo caso como parâmetro de entrada. Como parâmetro de saída, devolve uma lista de

casos recuperados, em que cada elemento da lista corresponde a um par caso-similaridade, onde é especificado cada caso e respetiva similaridade com o novo caso.

Algoritmo 8.6 – Fase de Recuperação

```
    Entrada: novoCaso
    Saída: listaCasos // Lista de casos recuperados e respetiva similaridade com o novo caso
1   inicializar(listaCasos);
2   simMin ← 0.70; // Similaridade mínima exigida para um caso ser recuperado
3   listaPreCasos ← consultarBaseCasos(novoCaso); // Efetuar uma pré-seleção de casos
4   // Percorrer os casos pré-selecionados
5   Para cada caso ∈ listaPreCasos Faz
6           // Calcular a medida de similaridade para cada um deles
7           sim ← calcularSimilaridade(caso, novoCaso);
8           Se sim >= simMin Então // Comparar com a similaridade mínima exigida
9                   listaCasos.adicionar(caso, sim); // Adicionar à lista de casos
10          Fim
11  Fim
12  Devolver listaCasos
```

O algoritmo começa por inicializar a lista de casos a devolver e por definir a similaridade mínima exigida para que um caso seja considerado suficientemente similar. Considera-se esta similaridade mínima igual a 0.70, num intervalo [0;1], o que significa que um caso necessita de ser 70% similar com o novo caso para ser selecionado. Esta similaridade mínima corresponde à soma dos pesos dos atributos *Njobs, ProbType* e *MultiLevel*, explicados mais à frente na equação (8.6).

Depois da inicialização das variáveis, o algoritmo começa por efetuar uma pré-seleção de casos, cujo objetivo é tirar partido das vantagens de uma consulta à Base de Casos para pré-selecionar apenas casos que possam ser considerados suficientemente similares, ou seja, para se ignorar casos que sejam muito pouco similares ao novo caso.

Tabela 8.19 – Comando SELECT

```
SELECT c.*, i.*, p.*
FROM Cbr c, Instance i, Params p
WHERE c.instanceID = i.ID and c.paramsID=p.id and i.NJobs between MinNjobs and MaxNjobs
```

A pré-seleção de casos a partir da base de casos é efetuada através do comando SELECT descrito na Tabela 8.19. São selecionados os casos que poderão ser candidatos à lista de casos a devolver. Assim, não serão selecionados casos muito pouco similares com o novo caso, uma vez que a sua análise implicaria um desperdício computacional.

Com este comando, considera-se uma restrição sobre o número de tarefas, pois este é o atributo mais importante, como se poderá constatar na descrição da medida de similaridade (equação (8.5)). O número de tarefas deve estar entre dois valores, um valor mínimo (equação (8.3)) e um valor máximo (equação (8.4)).

$$\mathrm{MinNjobs} = \mathrm{Njobs}_{\mathrm{NovoCaso}} * \mathrm{Peso}_{\mathrm{Njobs}} \tag{8.3}$$

$$\mathrm{MaxNjobs} = \frac{\mathrm{Njobs}_{\mathrm{NovoCaso}}}{\mathrm{Peso}_{\mathrm{Njobs}}} \tag{8.4}$$

O cálculo do valor mínimo é efetuado através do produto do número de tarefas pelo peso respetivo. Este peso está explicado na descrição da medida de similaridade (equação (8.6)), sendo cada peso um número real do intervalo [0;1]. Assim, o inverso é realizado para o cálculo do valor máximo, ou seja, é efetuado o quociente do valor do número de tarefas pelo respetivo peso.

Depois da pré-seleção de casos, estes são analisados de modo a comparar a respetiva similaridade com a similaridade mínima exigida. Assim, para cada caso da lista de casos pré-selecionados, é calculada a respetiva similaridade com o novo caso, e, se esta for melhor ou igual à similaridade mínima previamente especificada, o caso é adicionado à lista de casos a devolver, juntamente com o respetivo valor de similaridade. No final do algoritmo, esta lista de casos é devolvida.

Um dos aspetos mais importantes de um sistema de Raciocínio baseado em Casos é a medida de similaridade entre casos. Os atributos a considerar para a medida de similaridade estão guardados na tabela *Instance*: o número de tarefas (*NJobs*), o número de máquinas (*NMachines*), o tipo de problema (*ProblemType*), a característica multinível (*MultiLevel*), e o tempo de conclusão ótimo conhecido (*OptCmax*). Estes campos são ponderados diferenciadamente na medida de similaridade, definida na equação (8.5). A medida de similaridade é um valor entre zero (0) e um (1), correspondendo respetivamente a casos nada similares e casos iguais, sendo o resultado de uma soma ponderada das similaridades entre os diferentes atributos.

$$\begin{aligned}\mathrm{Sim} = {} & \mathrm{Peso}_{\mathrm{Njobs}} * \mathrm{Sim}_{\mathrm{Njobs}} + \mathrm{Peso}_{\mathrm{Nmachines}} * \mathrm{Sim}_{\mathrm{Nmachines}} + \mathrm{Peso}_{\mathrm{ProbType}} \\ & * \mathrm{Sim}_{\mathrm{ProbType}} + \mathrm{Peso}_{\mathrm{MultiLevel}} * \mathrm{Sim}_{\mathrm{MultiLevel}} + \mathrm{Peso}_{\mathrm{OptCmax}} \\ & * \mathrm{Sim}_{\mathrm{OptCmax}}\end{aligned} \tag{8.5}$$

Considerem-se os seguintes pesos para cada atributo:

$$\text{Peso}_{\text{Njobs}} = 0.5;\ \ \text{Peso}_{\text{NMachines}} = 0.25;\ \ \text{Peso}_{\text{ProbType}} = 0.15;$$
$$\text{Peso}_{\text{MultiLevel}} = 0.05;\ \text{Peso}_{\text{OptCmax}} = 0.05 \tag{8.6}$$

É dada maior importância ao número de tarefas e ao número de máquinas, pois estes são aqueles que melhor definem a dimensão de um problema, característica fundamental para a parametrização das Meta-heurísticas. Ainda assim, o número de tarefas é o principal atributo da dimensão de um problema, uma vez que define quantas tarefas serão processadas, enquanto o número de máquinas define como estas tarefas serão divididas para processamento. Assim, considerou-se um peso de 50% e de 25% para o número de tarefas e número de máquinas respetivamente. Estes dois atributos representam conjuntamente 75% da medida de similaridade. O tipo de problema, com 15%, apresenta alguma importância para a definição de parâmetros, uma vez que, por exemplo, um problema *Job-Shop* apresenta uma complexidade adicional quando comparado com um problema de máquina única. Considera-se que as características de multinível e de tempo de conclusão ótimo não são tão importantes como os restantes atributos, apresentando uma importância de 5% cada uma, mas revelam-se importantes para a verificação de casos muito similares. O atributo de tempo de conclusão ótimo foi considerado para ser possível saber se o novo caso já foi ou não anteriormente processado, pois, quando conhecida, esta é uma característica única de uma instância de problema.

A similaridade do número de tarefas (equação (8.7)) e a similaridade do número de máquinas (equação (8.8)) são calculadas da mesma forma, correspondendo à divisão do valor mais baixo pelo valor mais alto, sendo um valor no intervalo [0;1]. A similaridade do tipo de problema (equação (8.9)) e a similaridade da característica multinível (equação (8.10)) podem ser um (1) ou zero (0) caso os atributos sejam iguais ou diferentes, respetivamente. No caso do atributo do tempo de conclusão ótimo (equação (8.11)), a similaridade é calculada de forma idêntica aos atributos de número de tarefas e número de máquinas, caso os valores dos dois casos seja positivo. No caso de algum valor ser inferior a zero (0), isto significa que o valor não é conhecido, e nesse caso a similaridade é nula.

$$\text{Sim}_{\text{Njobs}} = \frac{\min(\text{Njobs}_1, \text{Njobs}_2)}{\max(\text{Njobs}_1, \text{Njobs}_2)} \tag{8.7}$$

$$\mathrm{Sim}_{\mathrm{Nmachines}} = \frac{\min(\mathrm{Nmachines}_1, \mathrm{Nmachines}_2)}{\max(\mathrm{Nmachines}_1, \mathrm{Nmachines}_2)} \tag{8.8}$$

$$\mathrm{Sim}_{\mathrm{ProbType}} = \begin{cases} 0, & \mathrm{ProbType}_1 \neq \mathrm{ProbType}_2 \\ 1, & \mathrm{ProbType}_1 = \mathrm{ProbType}_2 \end{cases} \tag{8.9}$$

$$\mathrm{Sim}_{\mathrm{MultiLevel}} = \begin{cases} 0, & \mathrm{MultiLevel}_1 \neq \mathrm{MultiLevel}_2 \\ 1, & \mathrm{MultiLevel}_1 = \mathrm{MultiLevel}_2 \end{cases} \tag{8.10}$$

$$\mathrm{Sim}_{\mathrm{OptCmax}} = \begin{cases} \dfrac{\min(\mathrm{OptCmax}_1, \mathrm{OptCmax}_2)}{\max(\mathrm{OptCmax}_1, \mathrm{OptCmax}_2)}, & \mathrm{OptCmax}_1 \geq 0 \text{ e } \mathrm{OptCmax}_2 \geq 0 \\ 0, & \mathrm{OptCmax}_1 < 0 \text{ ou } \mathrm{OptCmax}_2 < 0 \end{cases} \tag{8.11}$$

Na fase de Reutilização é selecionado, da lista de casos devolvida pela fase anterior, o caso mais similar ao novo caso, sendo a respetiva solução sugerida como uma possível solução para a resolução do mesmo. Assim, a Meta-heurística e os respetivos parâmetros usados para a resolução desse caso, muito similar, são copiados para a resolução do novo caso.

A lista de casos devolvida pela fase de Recuperação é o parâmetro de entrada do algoritmo da fase de Reutilização (Algoritmo 8.7). Como parâmetro de saída, devolve uma Meta-heurística com a respetiva parametrização. Inicialmente, são inicializadas três variáveis, nomeadamente uma para servir de comparação para determinar se dois casos são muito similares (*simMuitoSimilares*), outra para guardar a lista de melhores casos presentes na lista de casos recuperados (*listaMelhoresCasos*), e outra para servir de comparação na determinação dos melhores casos (*racioMinimo*).

A primeira verificação a efetuar é se a lista de casos recuperados está ou não vazia. Se estiver vazia, então não existem casos com similaridade superior à similaridade mínima especificada na fase de Recuperação. Isto significa que serão usados os parâmetros pré-definidos pelo utilizador/perito, funcionando como ponto de partida para a resolução de casos futuros com os atributos similares ao novo caso.

Algoritmo 8.7 – Fase de Reutilização

```
    // Lista de casos recuperados e respetiva similaridade com o novo caso
    Entrada: listaCasos
    Saída: melhorCaso // Melhor caso
1   simMuitoSimilares ← 0.95; // Similaridade para validar se dois casos são muito similares
2   // Rácio mínimo entre o OptCmax e o Cmax, para adicionar um caso à listaMelhoresCasos
3   racioMinimo ← 0.75;
4   inicializar(listaMelhoresCasos); // Lista de melhores casos
5   Se listaCasos = ∅ Então // Se listaCasos estiver vazia
6         melhorCaso ← aplicaParametrosPreDefinidos(); // aplicar parâmetros pré-definidos
7   Senão // Senão percorre a os casos da listaCasos
8         Para cada caso ∈ listaCasos Faz
9               // Comparar a similaridade de cada caso com a melhor até ao momento
10              Se comparaSims(caso, melhorCaso) >= simMuitoSimilares Então
11                    racio ← caso.OptCmax / caso.cmax;
12                    // Em casos muito similares, comparar o rácio com o racioMinimo
13                    Se racio >= racioMinimo Então
14                          listaMelhoresCasos.adicionar(caso); // adiciona à lista
15                    Senão
16                          // Calcular o rácio para o melhorCaso
17                          racioMelhor ← melhorCaso.OptCmax / melhorCaso.cmax;
18                          // Comparar os dois rácios, para se obter o melhor caso
19                          Se racio = racioMelhor Então // Rácios iguais
20                                // Necessário comparar tempos de execução
21                                Se caso.timeExec < melhorCaso.timeExec Então
22                                      melhorCaso ← caso; //atualizar melhorCaso
23                                Fim
24                          Senão
25                                Se racio > racioMelhor Então // Caso mais eficaz
26                                      melhorCaso ← caso; // atualizar melhorCaso
27                                Fim
28                          Fim
29                    Fim
30              Senão // Casos não muito similares, comparar as similaridades diretamente
31                    Se caso.sim > melhorCaso.sim Então // Caso mais similar
32                          melhorCaso ← caso; // atualizar melhorCaso
33                    Fim
34              Fim
35              Se listaMelhoresCasos <> ∅ Então // Se a listaMelhoresCasos tiver elementos
36                    // seleciona um caso aleatório para ser considerado como melhor
37                    indice ← random(0,listaMelhoresCasos.size());
38                    melhorCaso ← listaMelhorCasos.get(índice);
39              Fim
40        Fim
41  Fim
42  Devolver melhorCaso
```

Se a lista tiver elementos, será percorrida para se selecionar os casos mais eficazes ou, em contra partida, o caso mais similar, se não houver casos suficientemente eficazes. Neste ciclo são efetuadas algumas verificações de modo a determinar se existem casos muito similares entre si, para, em caso positivo, ser então possível a seleção dos melhores casos ou a seleção do caso mais eficaz-eficiente, através da comparação por tempo de conclusão e tempo de execução. Assim, a verificação dos casos muito similares é efetuada com recurso à equação (8.12), comparando-se o valor resultante com a variável de similaridade usada para o efeito (*simMuitoSimilares*). Para esta variável considerou-se o valor de 0.95, o que significa que dois casos são muito similares se o rácio entre as suas similaridades com o novo caso for superior a 95%.

$$\mathrm{ComparaSims}(\mathrm{Caso1}, \mathrm{Caso2}) = \frac{\min(\mathrm{Sim}_{\mathrm{Caso1}}, \mathrm{Sim}_{\mathrm{Caso2}})}{\max(\mathrm{Sim}_{\mathrm{Caso1}}, \mathrm{Sim}_{\mathrm{Caso2}})} \tag{8.12}$$

Quando existem casos muito similares, é necessário calcular o rácio (equação (8.13)) entre *OptCmax* e o *Cmax* de cada caso, para se proceder à sua comparação com o *racioMinimo*. Esta variável indica que um caso é considerado como um dos melhores se o rácio do seu *OptCmax* pelo seu *Cmax* for igual ou superior a 75%. Quando isso acontece, o caso é adicionado na *listaMelhoresCasos*.

$$\mathrm{Racio}_{\mathrm{Caso}} = \frac{\mathrm{CmaxOpt}_{\mathrm{Caso}}}{\mathrm{Cmax}_{\mathrm{Caso}}} \tag{8.13}$$

Quando o rácio de um caso não é superior ao *racioMinimo*, é necessário calcular o rácio entre o *OptCmax* e o *Cmax* do melhor caso até dado momento, para ser possível efetuar a comparação na determinação do melhor caso. No caso do rácio de um determinado caso ser igual ao rácio do *melhorCaso* (equação (8.14)), então os dois casos são igualmente eficazes, sendo necessário comparar os tempos de execução. Se o tempo de execução do caso for inferior ao do *melhorCaso* em dado momento, então o *melhorCaso* será atualizado, pois estamos perante um caso mais eficiente. Por outro lado, quando o rácio do caso é superior ao rácio do *melhorCaso*, então o *melhorCaso* será igualmente atualizado, pois estamos perante um caso mais eficaz. Assim, com o valor o rácio entre o *OptCmax* e o *Cmax* juntamente com o valor do tempo de execução é possível analisar o binómio eficiência-eficácia.

$$\text{Racio}_{\text{MelhorCaso}} = \frac{\text{OptCmax}_{\text{MelhorCaso}}}{\text{Cmax}_{\text{MelhorCaso}}} \tag{8.14}$$

Quando não existem casos muito similares, a escolha do melhor caso é efetuada através de comparação direta entre as similaridades dos vários casos da lista.

Após a lista de casos recuperados ter sido toda percorrida, e se a *listaMelhoresCasos* não estiver vazia, é então selecionado aleatoriamente um dos melhores casos. Deste modo garante-se a escolha de um dos melhores casos como *melhorCaso*, e não o melhor de todos. Isto revela-se importante para evitar estagnação e para evitar a escolha do mesmo caso demasiadas vezes, que desse modo não permitiria a evolução do sistema. Se fosse selecionado o melhor caso de entre todos, sempre que um novo caso fosse muito similar seria continuamente selecionado o mesmo caso anterior, a não ser que os novos casos obtivessem sempre novos resultados, o que não acontece devido à componente estocástica subjacente às Meta-heurísticas.

Se a *listaMelhoresCasos* estiver vazia, o *melhorCaso* é aquele que se revelou mais similar ,ou então é o caso que apresenta a melhor relação eficácia-eficiência de entre os casos mais similares.

Algoritmo 8.8 – Fase de Revisão

```
    Entrada: metaHeuristica, simMelhorCaso // Meta-heurística e similaridade do melhor caso
    Saída: metaHeuristicaRevista // Meta-heurística com perturbação nos parâmetros
1   inicializar(metaHeuristicaRevista);
2   simCredMin ← 0.95; // Variável auxiliar para comparação com a simMelhorCaso
3   credito ← 0; // Crédito global para ser utilizado na perturbação aos parâmetros
4   Se simCredMin < simMelhorCaso // Se a simMelhorCaso for superior à simCredMin
5           // Credito inicializado a 15, corresponde ao crédito mínimo
6           credito ← 10 + (1 - simCredMin) * 100;
7   Senão
8           // Credito inicializado em função da simMelhorCaso
9           credito ← 10 + (1 - simMelhorCaso) * 100;
10  Fim
11          // Calcular os créditos para a inclusão da perturbação em cada parâmetro
12          arrayCreditos ← devolverCreditos(credito, metaHeuristica);
13          // Atribuir a perturbação aos parâmetros, em função do arrayCreditos
14          metaHeuristicaRevista ← metaHeuristica.atualizarParametros(arrayCreditos);
15          Devolver metaHeuristicaRevista;
```

Na fase de Revisão (Algoritmo 8.8), procede-se a uma adaptação da solução sugerida pela fase de Reutilização. A utilização direta das soluções sugeridas leva

igualmente a que o sistema possa convergir para a estagnação e não consiga evoluir para a obtenção de melhores resultados. Assim, para escapar a soluções ótimas locais e estagnação do sistema, foi implementado um algoritmo que aplica alguma diversidade à solução sugerida pelo sistema de Raciocínio baseado em Casos. Esta diversidade passa pela aplicação de alguma perturbação aos parâmetros sugeridos.

Este algoritmo recebe dois parâmetros de entrada, nomeadamente a solução sugerida pela fase de Reutilização (i.e. a Meta-heurística sugerida com os respetivos parâmetros) e a similaridade do melhor caso. O algoritmo devolve a mesma Meta-heurística com os parâmetros revistos e alterados de modo a se tentar obter melhores resultados. São utilizadas duas variáveis importantes para a execução do algoritmo, o *credito* e a *simCredMin*, que representam, respetivamente, o crédito global a ser distribuído pelos parâmetros da Meta-heurística, para inclusão de perturbação, e uma variável auxiliar para garantir o crédito mínimo utilizado para a perturbação. Esta última é inicializada com o valor 0.95 (95%), que representa a similaridade máxima para a qual o crédito utilizado é inversamente proporcional. Isto significa que casos com similaridades superiores a 95% irão ter igual crédito global para ser aplicado na inserção de perturbação, sendo este crédito igual a 15, resultante da soma do valor 10 com a diferença da similaridade multiplicada por 100 (equação (8.15)).

$$\text{Credito} = 10 + (1 - \text{simCredMin}) * 100 \tag{8.15}$$

Assim, a segunda tarefa a realizar pelo algoritmo (a seguir à inicialização de variáveis) é verificar se a similaridade do caso mais similar é ou não superior a 95%. Se for, então o crédito é inicializado com o valor mínimo de 15, dado em função da variável *simCredMin*. Caso contrário, o crédito será inversamente proporcional à similaridade do melhor caso, isto é, do caso mais similar. Isto significa que quanto menos similar um caso for, mais perturbação será incluída nos parâmetros da Meta-heurística sugerida.

Depois de inicializado o crédito a ser distribuído pelos parâmetros, é efetuado o cálculo desta mesma distribuição, através da chamada ao método *devolverCreditos()*, descrito no Algoritmo 8.9. Com o crédito distribuído para aplicação nos parâmetros, procede-se então à atualização dos parâmetros da Meta-heurística, concluindo-se o algoritmo com a devolução da Meta-heurística atualizada. O algoritmo recebe dois parâmetros, o crédito global e a Meta-heurística com os parâmetros a atualizar, e devolve

um *array* com os créditos a serem utilizados em cada um dos parâmetros da Meta-heurística.

Algoritmo 8.9 – Método *devolverCreditos()*

```
    Entrada: credito, metaHeuristica // Crédito global e Meta-heurística
    Saída: arrayCreditos // Array com os créditos a ser aplicados a cada parâmetro
1   inicializar(arrayCreditos);
2   // Percorrer os parâmetros da Meta-heurística
3   Para cada parametro ∈ metaHeuristica.listaParametros Faz
4           // Enquanto o crédito global não for nulo
5           Se credito > 0 Então
6                       // Atribuir crédito ao parâmetro, no intervalo [0 ; credito/2]
7                       arrayCreditos[parametro] ← random(0, credito/2);
8                       // Descontar o crédito atribuído ao crédito global
9                       credito = credito - arrayCreditos[parametro];
10                      // Decidir o sinal do crédito
11                      Se random(0,1) = 1 Então
12                              // Se o valor for "verdadeiro" (1), o crédito fica negativo
13                              arrayCreditos[parametro] ← - arrayCreditos[parametro];
14                      Fim
15                  Senão
16                      // Se já não houver crédito, não é atribuído nenhum ao parâmetro
17                      arrayCreditos[parametro] ← 0;
18                  Fim
19          Fim
20          Devolver arrayCreditos;
```

Depois de inicializado o *array* de créditos, procede-se à atualização dos parâmetros, um a um. Só se poderá atribuir crédito a um parâmetro se existir crédito global disponível. Assim, essa é a primeira verificação a efetuar. Se houver crédito disponível então o valor de crédito atribuído ao parâmetro é um valor aleatório no intervalo [0 ; credito/2], sendo o máximo de crédito atribuído igual à metade do crédito global. Este valor aleatório é guardado no *array* e descontado ao crédito global. De seguida, é realizado um cálculo aleatório para se decidir se a perturbação será adicionada ou retirada ao parâmetro. Assim, se o valor for "*verdadeiro*" (1) então o crédito do parâmetro é colocado com sinal negativo, para resultar num decréscimo do valor do parâmetro, ao invés de um incremento. No final, o *array* dos créditos dos parâmetros é devolvido.

Nas equações (8.16) e (8.17) são apresentadas as fórmulas para atualização dos parâmetros das Meta-heurísticas. A primeira é respeitante à atualização de parâmetros com valores inteiros, enquanto a segunda diz respeito à atualização de parâmetros com vírgula flutuante.

$$\text{Parametro} = \text{Parametro} + round\left(\left(\text{Parametro} * \frac{\text{ArrayCreditos[Parametro]}}{100}\right), 0\right) \quad (8.16)$$

$$\text{Parametro} = \text{Parametro} + round\left(\left(\text{Parametro} * \frac{\text{ArrayCreditos[Parametro]}}{100}\right), 2\right) \quad (8.17)$$

Nos números inteiros é efetuado um arredondamento às unidades do produto do valor do parâmetro com o respetivo crédito, sendo este arredondamento somado (ou subtraído, dependendo do sinal do crédito) ao valor original. O mesmo se passa com os valores de vírgula flutuante, mas o arredondamento é realizado à segunda casa decimal. Com este procedimento, o sistema é capaz de introduzir perturbação nas soluções sugeridas, de modo a escapar à estagnação e a evoluir para melhores resultados.

Finalmente, depois de testar o novo caso no sistema, torna-se necessário armazenar a informação do novo caso resolvido na base de casos, através da fase de Retenção. Primeiro, são recolhidos os valores dos tempos de conclusão e de execução resultantes da execução do novo caso no Sistema Multiagente e de seguida criam-se os registos nas tabelas *Instance*, *Cbr*, *Params* e na tabela da Meta-heurística usada na resolução do novo caso, de modo a guardar a solução para uso posterior. Com esta fase, conclui-se o ciclo do sistema de Raciocínio baseado em Casos. Na próxima execução, o caso resolvido já estará disponível para ser usado na resolução de um novo caso.

8.4.2. Exemplo ilustrativo

Neste exemplo, pretende-se simular a chegada de um novo caso ao sistema e resolvê-lo através do ciclo de Raciocínio baseado em Casos. Para ilustrar o funcionamento do módulo de Raciocínio baseado em Casos, considere-se os dados da Tabela 8.20, que correspondem à instância a resolver.

Tabela 8.20 – Novo caso

NJobs	NMachines	ProbType	MultiLevel	OptCmax
10	5	Job-Shop	não	655

A base de casos é composta por 10 resoluções de diferentes instâncias (casos). Na Tabela 8.21 é possível visualizar os dados das instâncias resolvidas, os resultados obtidos pela execução e qual a Meta-heurística utilizada para a sua resolução.

Tabela 8.21 – Base de casos (tabelas *Instance, Cbr* e *Params)*

	Instance					Cbr		Params		
ID	**NJobs**	**NMachines**	**ProbType**	**Multi Level**	**OptCmax**	**Cmax**	**Time Exec**	**MH**	**InitSol**	**ObjFunc**
1	10	5	*Job-Shop*	não	666	753	7,64	*TS*	*SeqNivel*	CMax
2	20	10	*Job-Shop*	não	1292	1292	0,21	*SA*	*SeqNivel*	WT
3	15	5	*Job-Shop*	não	890	1150	27,41	*GA*	*SeqNivel*	CMax
4	10	10	*Job-Shop*	não	945	1144	2,94	*TS*	*SeqNivel*	WT
5	10	10	*Job-Shop*	não	784	1057	7,5	*GA*	*SeqNivel*	WT
6	30	10	*Job-Shop*	não	1784	2086	14,14	*TS*	*SeqNivel*	WT
7	50	10	*Job-Shop*	não	2924	3120	5,52	*SA*	*SeqNivel*	CMax
8	15	10	*Job-Shop*	não	935	1285	0,22	*SA*	*SeqNivel*	CMax
9	15	10	*Job-Shop*	não	1032	1464	29,8	*GA*	*SeqNivel*	CMax
10	10	10	*Job-Shop*	não	848	1035	2,02	*TS*	*SeqNivel*	WT

Para este exemplo foram utilizadas três Meta-heurísticas: Pesquisa Tabu (*Ts*), Algoritmos Genéticos (*Ga*) e *Simulated Annealing* (*Sa*). Os parâmetros usados na resolução de cada caso estão definidos na Tabela 8.22, Tabela 8.23, e Tabela 8.24.

Tabela 8.22 – Tabela *Ts*

ID	NeighGen	SubNeigh	TabuListLen	StopCrit
1	0,15	1,00	1	50
4	0,15	1,00	1	75
6	0,15	1,00	3	175
10	0,15	1,00	1	50

Tabela 8.23 – Tabela *Ga*

ID	InitPopGen	PopSize	NumGen	CrossRate	MutRate
3	0,15	1,00	125	0,75	0,01
5	0,15	1,00	100	0,75	0,01
9	0,15	1,00	150	0,75	0,01

Tabela 8.24 – Tabela *Sa*

ID	InitTemp	Alpha	NumItK	StopCrit
2	15	0,8	25	80
7	15	0,8	30	200
8	15	0,8	10	70

O ciclo de Raciocínio baseado em Casos é iniciado pela fase de Recuperação, sendo inicialmente efetuada uma pré-seleção de casos, delimitada pelos valores mínimo (equação (8.3)) e máximo (equação (8.4)) do número de tarefas. O comando de seleção de dados encontra-se descrito na Tabela 8.25.

$$\text{MinNjobs} = 10 * 0.5 = 5$$

$$\text{MaxNjobs} = \frac{10}{0.5} = 20$$

Tabela 8.25 – Comando SELECT

```
SELECT c.*, i.*, p.*
FROM Cbr c, Instance i, Params p
WHERE c.instanceID = i.ID and c.paramsID=p.id and i.NJobs between 5 and 20
```

Com base nesta seleção, é possível descartar, logo à partida, os casos que terão uma similaridade muito baixa com o novo caso. Estes casos, que não têm uma similaridade suficientemente alta para serem considerados são os números 6 e 7.

Assim, o passo seguinte consiste em calcular a similaridade que cada um destes casos tem para com o novo caso. Os casos que têm uma similaridade superior à similaridade mínima definida (70%) são selecionados para passarem à próxima fase. Na Tabela 8.26, é possível visualizar os valores de cada similaridade e os casos que passam para a fase seguinte encontram-se assinalados a verde.

Os casos recuperados da fase de Recuperação dão entrada na fase de Reutilização. Nesta fase, o objetivo é verificar que casos apresentaram melhor desempenho em termos de eficácia na resolução de problemas anteriores. Para isso, é calculado o rácio entre o tempo de conclusão (*Cmax*) e o tempo de conclusão ótimo (*OptCmax*). Os casos cujo rácio

seja superior ao rácio mínimo definido (75%) são selecionados para o passo seguinte. Na Tabela 8.27 é possível verificar os cálculos para cada caso. Os casos assinalados passam para o passo seguinte.

Tabela 8.26 – Fase de Recuperação

ID	SimNJobs	SimNMachines	SimProbType	SimMultiLevel	SimOptCmax	Similaridade
1	0,5	0,25	0,15	0,05	0,049	0,999
2	0,25	0,125	0,15	0,05	0,025	0,600
3	0,333	0,25	0,15	0,05	0,037	0,820
4	0,5	0,125	0,15	0,05	0,035	0,860
5	0,5	0,125	0,15	0,05	0,042	0,867
8	0,333	0,125	0,15	0,05	0,035	0,693
9	0,333	0,125	0,15	0,05	0,032	0,690
10	0,5	0,125	0,15	0,05	0,039	0,863

Tabela 8.27 – Exemplo ilustrativo de Raciocínio baseado em Casos - Fase de Reutilização

ID	Similaridade	OptCmax	Cmax	Rácio
1	0,999	666	753	0,884
3	0,820	890	1150	0,773
4	0,860	945	1144	0,826
5	0,867	784	1057	0,741
10	0,863	848	1035	0,819

Dos quatro casos reutilizados, é selecionado um de forma aleatória, pois não se pode considerar que um deles tenha uma clara vantagem sobre os outros. É importante voltar a salientar que, se fosse sempre selecionado o caso com maior rácio, o sistema poderia convergir para uma estagnação pois não é garantido que a técnica usada irá tirar melhores resultados. A diversidade pretendida por este módulo poderia ficar comprometida. Por outro lado, o mesmo se passa com a similaridade. Se fosse sempre selecionado o caso com maior similaridade, o sistema também poderia convergir para uma estagnação pois, a não ser que a similaridade fosse de 100%, não estamos perante a mesma instância. Mas, mesmo quando estamos perante a mesma instância, não é possível garantir que a técnica usada na resolução daquele caso é a mais adequada. Assim, deste modo, pretende-se garantir a reutilização de um caso que é suficientemente similar com o novo caso e que tem possibilidades de obter uma solução próxima do ótimo global.

Neste exemplo, o caso selecionado aleatoriamente foi o número 4, que foi resolvido com recurso à Pesquisa Tabu. Na Tabela 8.28 encontram-se os parâmetros da técnica e, na Tabela 8.29, a heurística de solução inicial e qual a função objetivo a minimizar.

Tabela 8.28 – Solução do melhor caso

ID	NeighGen	SubNeigh	TabuListLen	StopCrit
4	0,15	1,00	1	75

Tabela 8.29 – Solução inicial e função objetivo usadas pelo melhor caso

InitSol	ObjFunc
SeqNivel	WT

O caso número 4 passa assim para a fase de Revisão, onde o objetivo é afinar os parâmetros da Meta-heurística selecionada. Tal como já foi referido anteriormente, esta afinação é realizada através de um crédito global que é dependente da similaridade do caso para com o novo caso. Com base na equação (8.15) é possível verificar o cálculo deste crédito:

$$\text{Credito} = 10 + (1 - 0.86) * 100 = 24$$

Com base no crédito global, a perturbação associada a cada parâmetro depende da sua relevância na pesquisa do espaço de soluções. Assim, neste exemplo, considera-se que para a técnica Pesquisa Tabu, o parâmetro que apresenta maior influência na exploração do espaço de soluções é o número de iterações, seguido do afastamento máximo. Assim, para cada parâmetro é, aleatoriamente, atribuído um crédito até um máximo de metade do crédito atual, sendo a cada crédito atribuído um sinal positivo ou negativo

Na Tabela 8.30 estão apresentados os créditos atribuídos bem como os cálculos efetuados e o respetivo valor do parâmetro atualizado.

Tabela 8.30 – Fase de Revisão

Parâmetro	Crédito atribuído	Cálculo	Valor atualizado
StopCrit	-11	75 + 75 * (-11) / 100	67
NeighGen	5	0,15 + 0,15 * 5 / 100	0,16
SubNeigh	-4	1 + 1 * (-4) / 100	0,96
TabuListLen	2	1 + 1 * 2 / 100	1

Após a fase de Revisão, é necessário executar o Sistema Multiagente para retirar uma solução. Os resultados da execução dos parâmetros da Pesquisa Tabu na resolução do novo caso podem ser consultados na Tabela 8.31.

Tabela 8.31 – Resultados obtidos pela execução do novo caso

Cmax	TimeExec
763	2,71

Tabela 8.32 – Fase de Retenção (tabelas *Instance*, *Cbr* e *Params*)

	Instance					Cbr		Params		
ID	**NJobs**	**NMachines**	**ProbType**	**Multi Level**	**OptCmax**	**Cmax**	**Time Exec**	**MH**	**InitSol**	**ObjFunc**
1	10	5	*Job-Shop*	não	666	753	7,64	*TS*	*SeqNivel*	CMax
2	20	5	*Job-Shop*	não	1292	1292	0,21	*SA*	*SeqNivel*	WT
3	15	5	*Job-Shop*	não	890	1150	27,41	*GA*	*SeqNivel*	CMax
4	10	10	*Job-Shop*	não	945	1144	2,94	*TS*	*SeqNivel*	WT
5	10	10	*Job-Shop*	não	784	1057	7,5	*GA*	*SeqNivel*	WT
6	30	10	*Job-Shop*	não	1784	2086	14,14	*TS*	*SeqNivel*	WT
7	50	10	*Job-Shop*	não	2924	3120	5,52	*SA*	*SeqNivel*	CMax
8	15	10	*Job-Shop*	não	935	1285	0,22	*SA*	*SeqNivel*	CMax
9	15	10	*Job-Shop*	não	1032	1464	29,8	*GA*	*SeqNivel*	CMax
10	10	10	*Job-Shop*	não	848	1035	2,02	*TS*	*SeqNivel*	WT
11	10	5	*Job-Shop*	não	655	853	2,71	*TS*	*SeqNivel*	WT

Por último, é necessário guardar estes dados na base de casos para que possam ser futuramente utilizados na resolução de um caso similar. Esta última fase chama-se Retenção e tem o objetivo de criar um novo registo nas tabelas *Instance*, *Cbr* e *Params* (Tabela 8.32) bem como guardar os parâmetros na tabela *Ts* (Tabela 8.33).

Tabela 8.33 – Fase de Retenção (tabela *Ts*)

ID	NeighGen	SubNeigh	TabuListLen	StopCrit
1	0,15	1,00	1	50
4	0,15	1,00	1	75
6	0,15	1,00	3	175
10	0,15	1,00	1	50
11	0,16	0,96	1	67

8.5. Sumário

Neste capítulo foi dado um especial destaque à descrição do módulo de Auto-Otimização, desenvolvido e integrado do Sistema Multiagente *AutoDynAgents*, que surgiu como resposta à necessidade de resolução do problema autoparametrização das Meta-heurísticas. Nesse sentido, foram propostos e descritos dois mecanismos de aprendizagem baseados em: *Racing* e Raciocínio baseado em Casos. Estes mecanismos pretender dotar o sistema *AutoDynAgents* com a capacidade de autoparametrização das Meta-heurísticas na resolução do problema de escalonamento de tarefas em sistemas de produção.

Inicialmente, foi descrita a arquitetura para as abordagens de Aprendizagem para a Equipa (um só aprendiz) e Aprendizagem Concorrente (múltiplos aprendizes). Descreveu-se a arquitetura de uma forma global, com especificação da base de dados/casos, e foi apresentado o funcionamento geral e um exemplo ilustrativo de cada um dos módulos de aprendizagem.

Capítulo 9. Estudo Computacional

9.1. Introdução

Neste capítulo é apresentado e descrito o conjunto de testes ou experiências computacionais com vista à validação dos mecanismos de aprendizagem propostos.

A implementação do sistema *AutoDynAgents*, incluindo os mecanismos de aprendizagem, foi realizada na linguagem de programação Java, tendo sido usada a *framework Hibernate*[13] como camada de acesso à base de dados, esta implementada em H2[14]. A máquina usada para o estudo computacional foi uma HP Z400, com as seguintes características principais: processador Intel Xeon W3565 3.20 GHz, 6GB DDR3 de memória RAM, disco rígido Samsung HD103SJ de 1TB 7200rpm, e sistema operativo Microsoft Windows 7.

Neste capítulo, são apresentados os resultados computacionais obtidos no âmbito deste trabalho de doutoramento, tendo sido usadas instâncias de problemas de escalonamento académicos (*Job-Shop*) retirados da *OR-Library* (Beasley, 1990). A Tabela 9.1 contém as siglas de todos os autores, para uma melhor identificação das instâncias.

Tabela 9.1 – Autores e respetiva sigla das instâncias da *OR-Library* (Beasley, 1990)

Autor	Sigla
Fisher e Thompson (1963)	FT
Lawrence (1984)	La
Adams *et al.* (1988)	ABZ
Applegate e Cook (1991)	ORB
Storer *et al.* (1992)	SWV
T. Yamada e Nakano (1992)	YN
Taillard (1993)	Tai

As instâncias usadas ao longo do estudo computacional estão descritas na Tabela 9.2. Para cada uma é possível, através da sigla, verificar o autor e também a respetiva dimensão. Por exemplo, a instância “La05 10x5” significa que foi proposta em (Lawrence, 1984), é a 5ª da lista, com 10 tarefas executadas em 5 máquinas. Para uma melhor

[13] http://www.hibernate.org
[14] http://www.h2database.com

compreensão, dividiram-se as instâncias pelo número de tarefas, pois a dimensão e a inerente complexidade apresentam influência na análise das técnicas.

Tabela 9.2 – Instâncias usadas nos estudos computacionais

Número de tarefas	Instâncias de inicialização dos estudos	Instâncias de teste dos estudos
10	FT10 10x10 La01 10x5 La05 10x5 La17 10x10 ABZ510x10	ABZ6 10x10 la02 10x5 La04 10x5 La16 10x10 La20 10x10 ORB01 10x10
15	La06 15x5 La10 15x5 La21 15x10 La25 15x10 La40 15x15	La07 15x5 La08 15x5 La09 15x5 La22 15x10 La23 15x10 La38 15x15
20	FT20 20x5 La11 20x5 La29 20x10 SWV6 20x15 YN1 20x20	ABZ8 20x15 La13 20x5 La26 20x10 SWV01 20x10 SWV08 20x15 YN2 20x20
30	La31 30x10 La33 30x10 La35 30x10 Tai1 30x15 Tai3 30x20	La32 30x10 La34 30x10 Tai2 30x15 Tai4 30x20 Tai11 30x15 Tai12 30x20
50	SWV11 50x10 SWV15 50x10 SWV17 50x10 Tai5 50x15 Tai7 50x20	SWV12 50x10 SWV13 50x10 SWV14 50x10 SWV16 50x10 Tai6 50x15 Tai8 50x20

Todos os resultados obtidos correspondem à minimização do tempo de conclusão (*Cmax*). Cada instância foi executada 5 vezes, e calculou-se a média dos valores obtidos. De modo a normalizar os valores, é usado o quociente entre o valor ótimo e o valor médio de *Cmax* (equação (9.1)), para estimar o desvio do valor obtido em relação ao valor da solução ótima referenciada na literatura.

$$q = 1 - \frac{\mathrm{OptCmax}}{\mathrm{Cmax}} \tag{9.1}$$

Assim, em vez de uma comparação direta entre valores obtidos, é comparada a variação do valor médio de *Cmax* em relação ao ótimo, o que é particularmente vantajoso quando estamos perante valores de *Cmax* diferentes. Considere-se, por exemplo, duas instâncias, cujos valores ótimos são 50 e 80 respetivamente. Se obtivermos o valor 60 para a primeira instância e o valor 95 para a segunda, é possível concluir que foi obtido um valor mais próximo do ótimo na segunda instância, pois o quociente da primeira é 0,167 e o da segunda é 0,158.

O objetivo do estudo computacional é validar se existe ou não vantagem na utilização dos mecanismos de aprendizagem propostos no desempenho do sistema AutoDynAgents. Encontra-se dividido em cinco fases:

1. Estudo preliminar entre Aprendizagem para a Equipa e Aprendizagem Concorrente (secção 9.2);
2. Resultados computacionais previamente obtidos, sem incorporação de qualquer mecanismo de aprendizagem, resultante da análise de desempenho do sistema AutoDynAgents, para dar suporte ao estudo computacional (subsecção 9.3.1);
3. Módulo de *Racing* proposto (subsecção 9.3.2), dividido em inicialização do estudo e discussão dos resultados computacionais obtidos;
4. Módulo de Raciocínio baseado em Casos (subsecção 9.3.3), dividido em inicialização do estudo e discussão dos resultados;
5. Abordagem híbrida baseada em *Racing* e Raciocínio baseado em Casos (subsecção 9.3.4), onde a inicialização corresponde aos resultados obtidos pelo módulo de *Racing,* de forma independente. Os resultados computacionais obtidos através da introdução do Raciocínio baseado em Casos são discutidos.

9.2. Aprendizagem para a Equipa *vs.* Aprendizagem Concorrente

As abordagens descritas ao longo deste trabalho são baseadas na abordagem de Aprendizagem para a Equipa, pois só existe um aprendiz envolvido, que tenta melhorar o desempenho para todo o Sistema Multiagente, tal como descrito na secção 8.2. No entanto, para validar qual as duas abordagens a utilizar, foi realizado um estudo preliminar do

desempenho das abordagens de Aprendizagem para a Equipa e Aprendizagem Concorrente, na resolução de um conjunto de instâncias do problema de escalonamento de tarefas em sistemas de produção. Este estudo encontra-se publicado em (Pereira *et al.*, 2012; Pereira *et al.*, 2013b).

Comparando as duas abordagens, da análise da Tabela 9.3 e da Figura 9.1, é possível evidenciar a vantagem da abordagem de Aprendizagem para a Equipa na minimização de *Cmax*, quando comparado com os resultados da abordagem de Aprendizagem Concorrente.

Tabela 9.3 – Análise descritiva do quociente dos valores médios da execução do sistema com Aprendizagem para a Equipa e Aprendizagem Concorrente

	Valor mínimo	Valor máximo	Média	Desvio padrão
Aprendizagem para a Equipa	0,00	0,30	0,1620	0,08026
Aprendizagem Concorrente	0,14	0,56	0,3185	0,12111

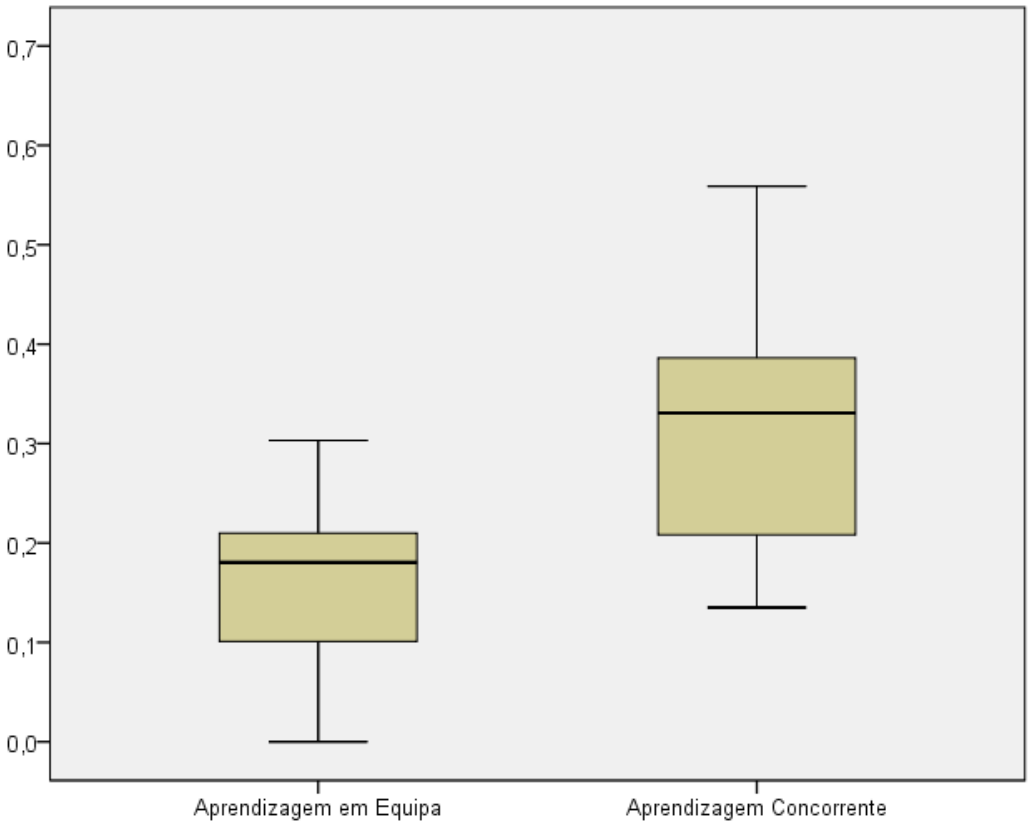

Figura 9.1 – Comparação do quociente dos valores médios da execução do sistema com Aprendizagem para a Equipa e Aprendizagem Concorrente

Com base no Teorema do Limite Central, que afirma que estamos perante uma distribuição normal para um tamanho da amostra $n \geq 30$. Neste caso pode ser usado o teste *t* de Student (Box, 1987), considerado o teste mais poderoso para analisar duas amostras (Conover, 1999). Analisando a significância estatística destes resultados (Tabela 9.4), com os valores $t(29) = -9{,}048; p < 0{,}05$ obtidos através do teste *t* de Student, pode-se afirmar, com um grau de confiança de 95%, que existem diferenças estatisticamente significativas entre as duas abordagens. É possível concluir que o sistema AutoDynAgents apresenta um melhor desempenho na resolução de *Cmax* com abordagem baseada em Equipa

Tabela 9.4 – Resultado do teste *t* de Student para amostras emparelhadas: Aprendizagem para a Equipa *vs.* Aprendizagem Concorrente

Média da diferença	Desvio padrão da diferença	*t*	Graus de liberdade	*p_value*
- 0,15657	0,09477	-9,048	29	0,000

Justifica-se assim a adoção da abordagem de Aprendizagem para a Equipa na implementação dos módulos de *Racing* e de Raciocínio baseado em Casos, permitindo concluir acerca da vantagem estatisticamente significativa da abordagem em Equipa quando comparada com a abordagem Concorrente.

9.3. Estudo dos Módulos de Aprendizagem

Nesta secção, é efetuada a validação do desempenho das três formas de funcionamento do módulo de Auto-Otimização, tal como descrito na secção 8.2, nomeadamente: a abordagem baseada em *Racing*, a abordagem de Raciocínio baseado em Casos, e a abordagem híbrida *Racing* + Raciocínio baseado em Casos.

9.3.1. Resultados Prévios

De modo a ser possível ter um ponto de referência interno da melhoria do sistema com a incorporação dos mecanismos de aprendizagem, foi necessário compilar os resultados previamente obtidos pelo sistema AutoDynAgents sem aprendizagem, resultantes da execução das técnicas de otimização. Estes resultados foram usados previamente em outros estudos e encontram-se publicados em (Madureira *et al.*, 2014; Madureira *et al.*, 2013c; Madureira *et al.*, 2011b; Pereira e Madureira, 2013; Pereira *et al.*, 2013a).

O resumo dos resultados obtidos previamente está ilustrado na Figura 9.2 e descrito na Tabela 9.5. Analisando estes valores, não é possível verificar uma diferença significativa entre o desempenho das técnicas, mas evidencia-se uma ligeira vantagem no uso de Pesquisa Tabu (TS) e Algoritmos Genéticos (GA) em relação a todas as outras, uma vez que apresentam uma média e um desvio padrão inferiores. Esta conclusão pode ser suportada pela mediana, medidas de dispersão e variabilidade.

Tabela 9.5 – Análise descritiva do quociente dos valores médios da execução das Meta-heurísticas, antes da incorporação dos métodos de aprendizagem

Meta-heurística	Valor mínimo	Valor máximo	Média	Desvio padrão
TS	0,05	0,51	0,3365	0,12498
GA	0,07	0,49	0,3327	0,10754
SA	0,05	0,58	0,3885	0,15371
ACO	0,08	0,62	0,4238	0,13956
PSO	0,07	0,61	0,3759	0,15305
ABC	0,07	0,70	0,4176	0,16329

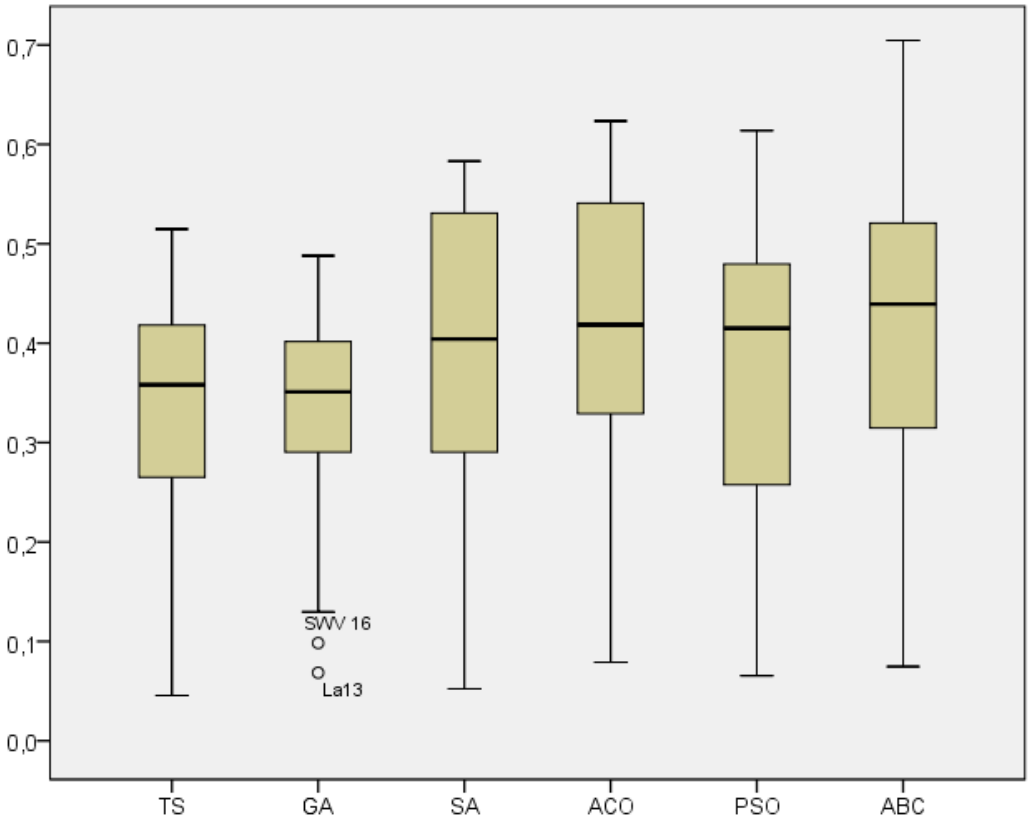

Figura 9.2 – Quociente dos valores médios da execução das Meta-heurísticas, antes da incorporação de qualquer método de aprendizagem

Não sendo um dos objetivos deste capítulo, mas de modo a completar este estudo computacional, é possível analisar também a eficiência de cada técnica. Na Tabela 9.6 é possível observar os tempos de execução médios de cada Meta-heurística, categorizados por número de tarefas. Genericamente, é possível identificar o *Simulated Annealing* como a técnica mais eficiente e a Otimização por Colónia de Formigas como a técnica menos eficiente. Como seria de esperar, evidencia-se uma relação de proporcionalidade direta do tempo de execução com o número de tarefas, o que revela que é necessário mais tempo para concluir uma execução numa instância de 50 tarefas do que numa instância de 20 ou 30. No entanto, dependendo da técnica, não se notam diferenças em instâncias mais pequenas (e.g. *Simulated Annealing* ou Algoritmos Genéticos).

Tabela 9.6 – Tempos de execução médios das Meta-heurísticas, antes da incorporação dos métodos de aprendizagem (em segundos)

Número de tarefas	TS	GA	SA	ACO	PSO	ABC
10	7,05	1,39	0,17	0,55	1,16	1,44
15	1,37	1,21	0,21	4,99	2,93	5,00
20	3,44	1,27	0,26	12,19	4,06	3,25
30	13,41	2,13	2,00	49,98	21,45	7,48
50	98,88	46,12	3,65	255,35	99,72	10,61

9.3.2. Abordagem baseada em *Racing*

O objetivo da abordagem baseada em *Racing* passa por obter uma parametrização adequada para as Meta-heurísticas, a partir de um conjunto inicial de combinações de parâmetros.

A inicialização do estudo de *Racing* consiste em criar conjuntos de combinações de parâmetros para cada Meta-heurística, de modo a poder evoluir ao longo das instâncias de treino e assim obter a parametrização mais adequada para a técnica em questão. É importante referir que, para cada técnica, se dividem os testes por número de tarefas, i.e. cada Meta-heurística é testada em instâncias de diferente dimensão, de modo a ser possível obter diferentes parâmetros para cada par {Meta-heurística; número de tarefas}.

Os valores para as combinações de parâmetros são apresentados na Tabela 9.7, Tabela 9.8, Tabela 9.9, Tabela 9.10, Tabela 9.11, e Tabela 9.12. Há parâmetros que permanecem inalterados durante o estudo, de forma a torná-lo exequível, pois considera-se que não tenham influência no desempenho como os restantes.

Tabela 9.7 – Valores para os parâmetros da Pesquisa Tabu, no estudo de *Racing*

MH	Número de tarefas	NeighGen	SubNeigh	TabuList Len	StopCrit
TS	10	15%	100%	1 2 3	50 75 100 125 150
	15				
	20			2 3 4	100 125 150 175 200
	30			3 4 5	150 175 200 225 250
	50			3 5 7	200 250 300 350 400

Para cada corrida são então criadas combinações de parâmetros para, no final, ser selecionada a melhor do estudo. Assim, por exemplo, para a Pesquisa Tabu, há 15 combinações de parâmetros (1 x 1 x 3 x 5) para cada dimensão. Estas combinações são testadas ao longo de 5 instâncias (ver Tabela 9.2), tal como mostrado na subsecção 8.3.2. Por outro lado, para a Otimização por Colónia de Formigas existem 36 combinações de parâmetros em cada corrida/instância. De modo a simplificar o estudo e manter a coerência, considerou-se o mesmo número de combinações de parâmetros candidatos para cada Meta-heurística, nas diferentes dimensões de instâncias do problema. No entanto, como o número de parâmetros de cada Meta-heurística difere, considerou-se diferentes números de candidatos entre Meta-heurísticas.

Tabela 9.8 – Valores para os parâmetros dos Algoritmos Genéticos, no estudo de *Racing*

MH	Número de tarefas	InitPop Gen	PopSize	NumGen	CrossRate	MutRate
GA	**10**	15%	100%	50 75 100 125 150	65% 75% 85%	1%
	15					
	20			100 125 150 175 200		
	30			150 175 200 225 250		
	50		35%	200 250 300 350 400		

Tabela 9.9 – Valores para os parâmetros da Colónia de Abelhas Artificiais, no estudo de *Racing*

MH	Número de tarefas	PopSize Sn	MaxFail	NumCycles
ABC	**10**	50 75 100	750 1000 1250	2000 2500 3000
	15	75 100 125	1750 2000 2250	3000 3500 4000
	20			3500 4000 4500
	30	100 125 150		4000 4500 5000
	50	125 150 175	2000 2250 2500	4000 4500 5000

Tabela 9.10 – Valores para os parâmetros do *Simulated Annealing*, no estudo de *Racing*

MH	Número de tarefas	InitTemp	Alpha	Num ItK	StopCrit
SA	10	15	80%	5 10 15	20 30 40 50
	15			10 15 20	40 50 60 70
	20			15 20 25	50 60 70 80
	30			20 25 30	60 70 80 90
	50			25 30 35	100 150 200 250

Tabela 9.11 – Valores para os parâmetros da Otimização por Colónia de Formigas, no estudo de *Racing*

MH	Número de tarefas	NumCol	Num Ants	Evap Rate	Alpha	Beta	StopCrit
ACO	10	1	20 40 60	80%	1 2	1 2	75 100 125
	15		30 50 70				100 150 200
	20						
	30						200 250 300
	50		50 75 100				300 350 400

Tabela 9.12 – Valores para os parâmetros do Particle Swarm Optimization, no estudo de *Racing*

MH	Número de tarefas	Num Part	Num It	Low Limit	Up Limit	Min Inertia	Max Inertia	C1	C2	Min Veloc	Max Veloc
PSO	10	20	1000	0	4	40%	95%	2	2	-4	4
		25	1250								
		30	1500								
		35	1750								
	15	25	1250								
		35	1500								
		45	1750								
		55	2000								
	20	25	1500								
		50	1750								
		75	2000								
		100	2250								
	30	50	2000								
		75	2500								
		100	3000								
		125	3500								
	50	75	3000								
		100	3500								
		125	4000								
		150	4500								

Seguindo esta metodologia, identificam-se 25 instâncias onde é efetuado o estudo, cinco por cada dimensão de número de tarefas. Como neste trabalho se consideraram 6 Meta-heurísticas, o objetivo passou por configurar 30 pares {Meta-heurística; dimensão}. Sendo que, cada Meta-heurística tem associado um número diferente de parâmetros (resultando em diferentes números de combinações de parâmetros em cada corrida). Considerou-se um total de 375 execuções por dimensão de número de tarefas, o que, multiplicando por 5 dimensões (10, 15, 20, 30, e 50 tarefas), dá um total de 1875 execuções no final do estudo.

No final deste estudo, foi possível identificar parâmetros diferentes daqueles que deram origem aos resultados prévios. Por se considerar terem uma maior relevância para o estudo de *Racing* + Raciocínio baseado em Casos, estes parâmetros são apenas apresentados na subsecção 9.3.4.

Após a obtenção da melhor combinação de parâmetros para cada par {Meta-heurística; número de tarefas}, foi possível retirar resultados para as instâncias de teste, de modo a analisar o desempenho da abordagem baseada em *Racing* e concluir quanto à existência, ou não, de vantagem significativa, comparando com os resultados obtidos previamente. Pretende-se comparar os resultados obtidos pelo módulo de *Racing* com os resultados prévios (secção 9.3.1), de modo a obter informação sobre a vantagem na utilização do módulo de aprendizagem baseado em *Racing*.

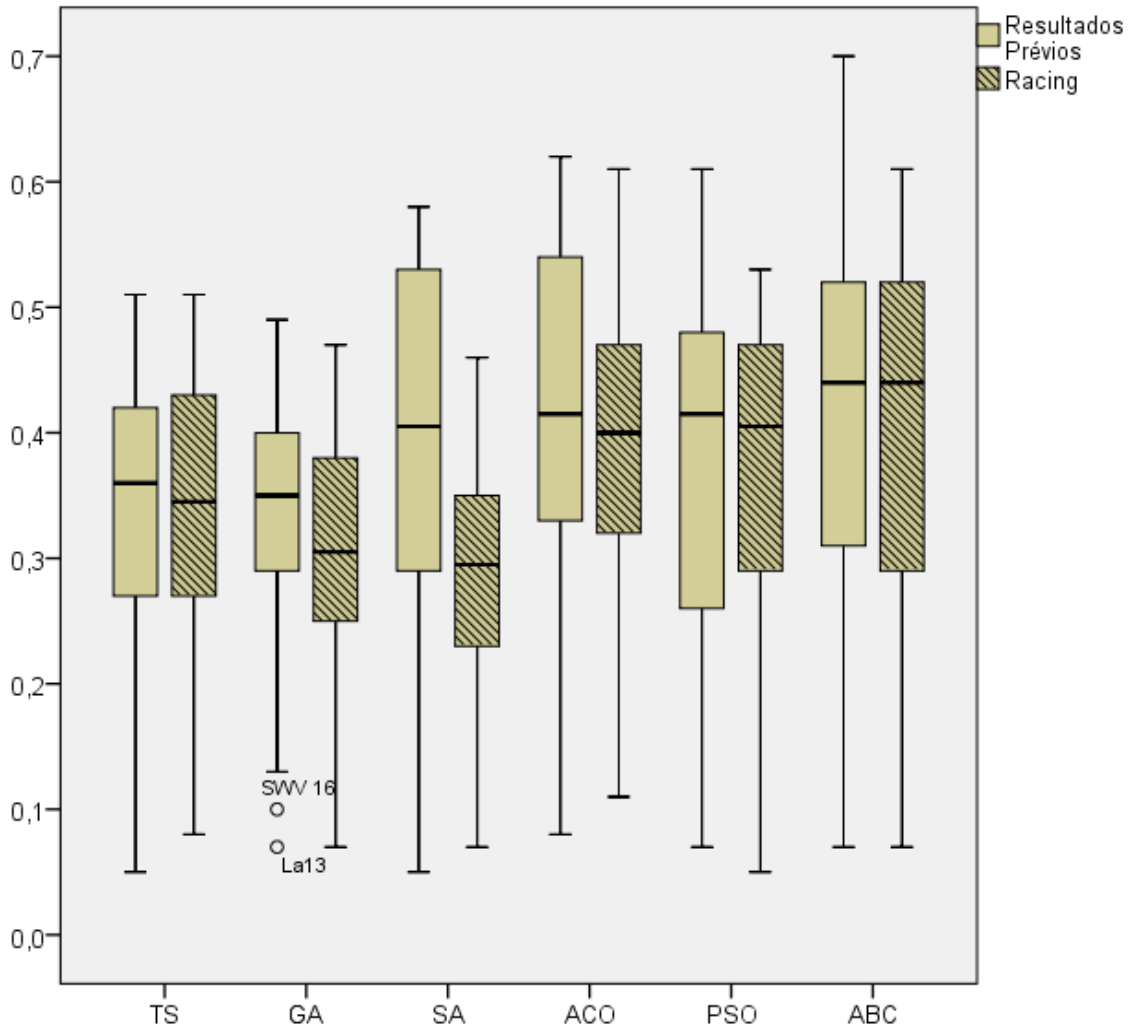

Figura 9.3 – Comparação do quociente dos valores médios da execução das Meta-heurísticas, entre os resultados prévios e os resultados de *Racing*

O gráfico da Figura 9.3 permite analisar a mediana e dispersão dos dados sintetizados por Meta-heurística sem e com aprendizagem por *Racing*. A Tabela 9.13 permite analisar os valores mínimos e máximos, a média e o desvio padrão desses mesmos dados. A partir da análise estatística dos resultados obtidos, é possível verificar vantagem

no uso de *Racing*, principalmente nos Algoritmos Genéticos e no *Simulated Annealing*, ao analisar a média dos valores. Analisando a dispersão dos resultados, é possível notar uma melhoria do desvio padrão em todas as técnicas. É possível assim afirmar que os parâmetros resultantes do estudo de *Racing* melhoraram os resultados obtidos, comparativamente com os resultados prévios associados ao sistema *AutoDynAgents* sem aprendizagem.

Tabela 9.13 – Análise descritiva do quociente dos valores médios da execução das Meta-heurísticas após o estudo de *Racing*

MH	Resultados	Valor mínimo	Valor máximo	Média	Desvio padrão
TS	Prévios	0,05	0,51	0,3365	0,12498
	Racing	0,08	0,51	0,3334	0,12226
GA	Prévios	0,07	0,49	0,3327	0,10754
	Racing	0,07	0,47	0,3025	0,10654
SA	Prévios	0,05	0,58	0,3885	0,15371
	Racing	0,07	0,46	0,2903	0,10556
ACO	Prévios	0,08	0,62	0,4238	0,13956
	Racing	0,11	0,61	0,3892	0,11597
PSO	Prévios	0,07	0,61	0,3759	0,15305
	Racing	0,05	0,53	0,3669	0,13401
ABC	Prévios	0,07	0,70	0,4176	0,16329
	Racing	0,07	0,61	0,4019	0,15480

De forma a poder analisar os resultados através do teste *t* de Student, normalizaram-se as amostras para ser possível comparar as abordagens diretamente de uma forma global. Esta normalização foi realizada através do cálculo do quociente dos valores médios (Figura 9.4). Analisando a significância estatística destes resultados, e observando os valores $t(29) = 4{,}740; p < 0{,}05$ na Tabela 9.14, pode-se afirmar, com um grau de confiança de 95%, que existem diferenças estatisticamente significativas entre os resultados obtidos inicialmente e os resultados obtidos pela abordagem baseada em *Racing*, permitindo concluir quanto à existência de vantagem estatisticamente significativa da abordagem baseada em *Racing* no desempenho do sistema *AutoDynAgents*.

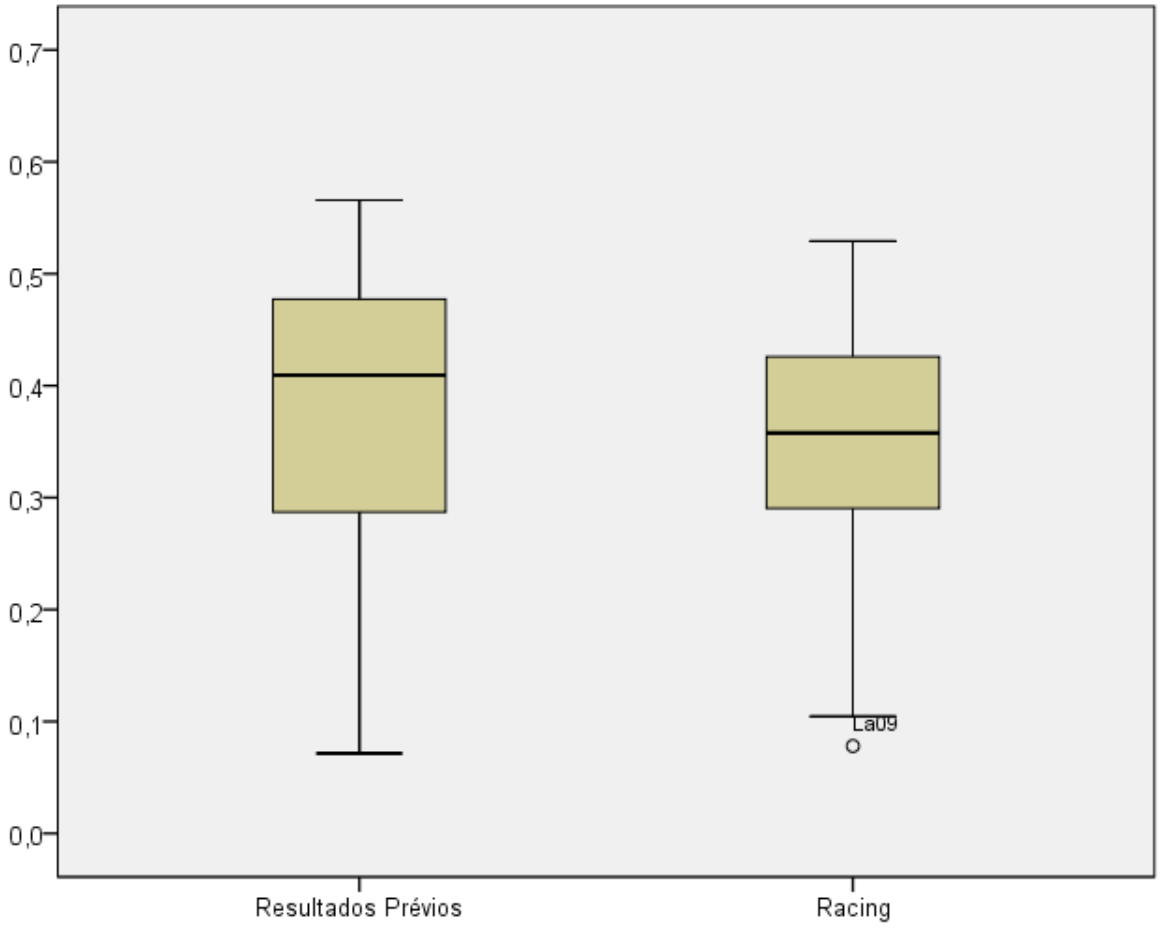

Figura 9.4 – Comparação do quociente dos valores médios da execução do sistema, entre os resultados prévios e os resultados de *Racing*

Tabela 9.14 – Resultado do teste t de Student para amostras emparelhadas: Resultados prévios vs. *Racing*

	Média da diferença	**Desvio padrão da diferença**	***t***	**Graus de liberdade**	***p_value***
Prévios *vs.* *Racing*	0,03182	0,03677	4,740	29	0,000

De modo a complementar o estudo, apresentam-se também os tempos de execução médios destes resultados na Tabela 9.15. A tendência mantém-se, mas evidenciam-se também melhorias em relação aos resultados prévios, como ilustrado na Figura 9.5. Assim, além de melhorar a eficácia dos resultados, a abordagem baseada em *Racing* permitiu melhorar também a eficiência do sistema.

Tabela 9.15 – Tempos de execução médios das Meta-heurísticas após o estudo de *Racing* (em segundos)

Número de tarefas	TS	GA	SA	ACO	PSO	ABC
10	1,54	0,27	0,22	0,32	0,93	1,18
15	0,49	0,35	0,25	1,15	1,24	1,89
20	2,06	0,62	0,39	2,17	4,71	2,49
30	10,31	1,99	0,56	29,43	20,60	3,80
50	41,24	31,32	0,71	144,42	47,53	10,33

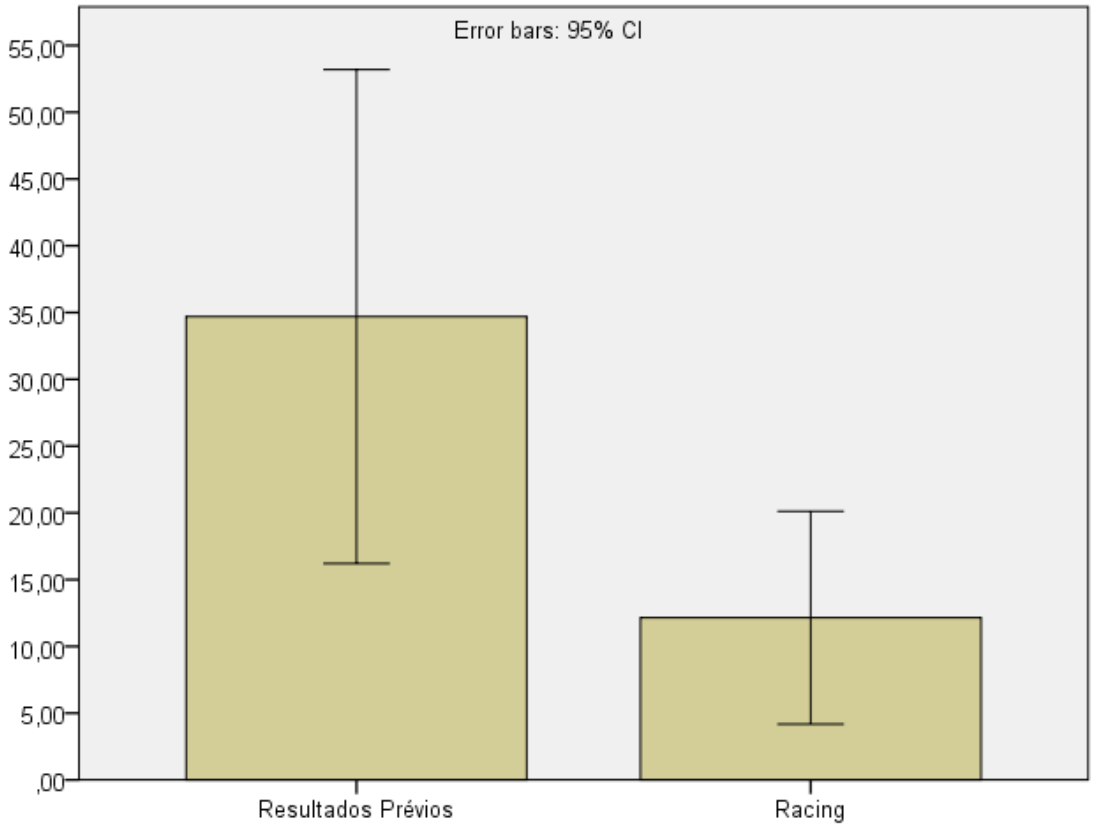

Figura 9.5 – Comparação dos tempos de execução médios, entre os resultados prévios e os resultados de *Racing*

9.3.3. Abordagem de Raciocínio baseado em Casos

A análise de desempenho do módulo de Raciocínio baseado em Casos foi dividida em duas fases: primeiro, a base de casos é inicializada com base nos resultados prévios, para as instâncias de inicialização (ver Tabela 9.2); depois são retirados resultados para as instâncias de teste, da mesma forma que foram retirados para os resultados prévios e para o estudo de *Racing*. Finalmente, os resultados são comparados entre si.

A base de casos foi inicializada a partir da média dos valores obtidos com os parâmetros prévios na execução das instâncias de inicialização. Assim, para cada instância, as diferentes Meta-heurísticas foram executadas 5 vezes, e calculada a média dos resultados. Como são 6 Meta-heurísticas, foi necessário efetuar 30 execuções por instância. Além disso, uma vez que existem 25 instâncias, foi necessário executar o sistema 750 vezes (6 Meta-heurísticas x 5 execuções x 25 instâncias). No entanto, como são inseridos os valores médios na base de casos, isto corresponde a 150 casos iniciais.

Como parâmetros iniciais, consideram-se os definidos na Tabela 9.16, Tabela 9.17, Tabela 9.18, Tabela 9.19, Tabela 9.20, e Tabela 9.21.

Tabela 9.16 – Valores para os parâmetros da Pesquisa Tabu, no estudo de Raciocínio baseado em Casos

MH	Número de tarefas	NeighGen	SubNeigh	TabuList Len	StopCrit
TS	10	15%	100%	1	100
	15			3	
	20				
	30				200
	50				350

Tabela 9.17 – Valores para os parâmetros dos Algoritmos Genéticos, no estudo de Raciocínio baseado em Casos

MH	Número de tarefas	InitPop Gen	PopSize	NumGen	CrossRate	MutRate
GA	10	15%	100%	100	75%	1%
	15					
	20					
	30			200		
	50		35%	250		

Tabela 9.18 – Valores para os parâmetros de *Simulated Annealing*, no estudo de Raciocínio baseado em Casos

MH	Número de tarefas	InitTemp	Alpha	Num ItK	StopCrit
SA	10	15	80%	15	35
	15				50
	20				
	30			25	100
	50			30	200

Tabela 9.19 – Valores para os parâmetros de Otimização por Colónia de Formigas, no estudo de Raciocínio baseado em Casos

MH	Número de tarefas	NumCol	Num Ants	Evap Rate	Alpha	Beta	StopCrit
ACO	10	1	25	80%	1	1	100
	15		50				250
	20						
	30						300
	50						350

Tabela 9.20 – Valores para os parâmetros de *Particle Swarm Optimization*, no estudo de Raciocínio baseado em Casos

MH	Número de tarefas	Num Part	Num It	Low Limit	Up Limit	Min Inertia	Max Inertia	C1	C2	Min Veloc	Max Veloc
PSO	10	25	1000	0	4	40%	95%	2	2	-4	4
	15	35	2000								
	20										
	30	75	3000								
	50	100	4000								

Tabela 9.21 – Valores para os parâmetros da Colónia de Abelhas Artificiais, no estudo de Raciocínio baseado em Casos

MH	Número de tarefas	PopSize Sn	MaxFail	NumCycles
ABC	**10**	50	1000	3000
	15	100	2000	4500
	20			
	30			
	50			

Após o preenchimento da base de casos com os dados iniciais, foi possível executar o sistema com o módulo de Raciocínio baseado em Casos para assim, retirar resultados computacionais. Como já referido, estes correspondem à minimização de *Cmax* e cada instância de teste foi executada cinco vezes, para se retirar a média dos valores obtidos. Para normalizar os valores, é usado o quociente entre o valor ótimo e o valor médio de *Cmax*, para estimar o desvio do valor obtido em relação ao ótimo.

Tabela 9.22 – Análise descritiva do quociente dos valores médios da execução do sistema após o estudo de Raciocínio baseado em Casos

	Valor mínimo	Valor máximo	Média	Desvio padrão
Resultados prévios	0,07	0,57	0,3792	0,13217
Racing	0,08	0,53	0,3474	0,11834
Raciocínio baseado em Casos	0,09	0,50	0,3388	0,10736

Comparando os resultados do módulo de Raciocínio baseado em Casos com os resultados prévios (Figura 9.6 e Tabela 9.22), é possível verificar quanto à vantagem, tanto na média como no desvio padrão. Foi usado o teste t de Student para analisar a significância estatística destes resultados, através da observação dos valores $t(29) = 3{,}963; p < 0{,}05$ na Tabela 9.23, pode-se afirmar, com um grau de confiança de 95%, que existem diferenças estatisticamente significativas entre os resultados obtidos inicialmente e os resultados obtidos pela abordagem de Raciocínio baseado em Casos, permitindo concluir quanto à vantagem na sua utilização.

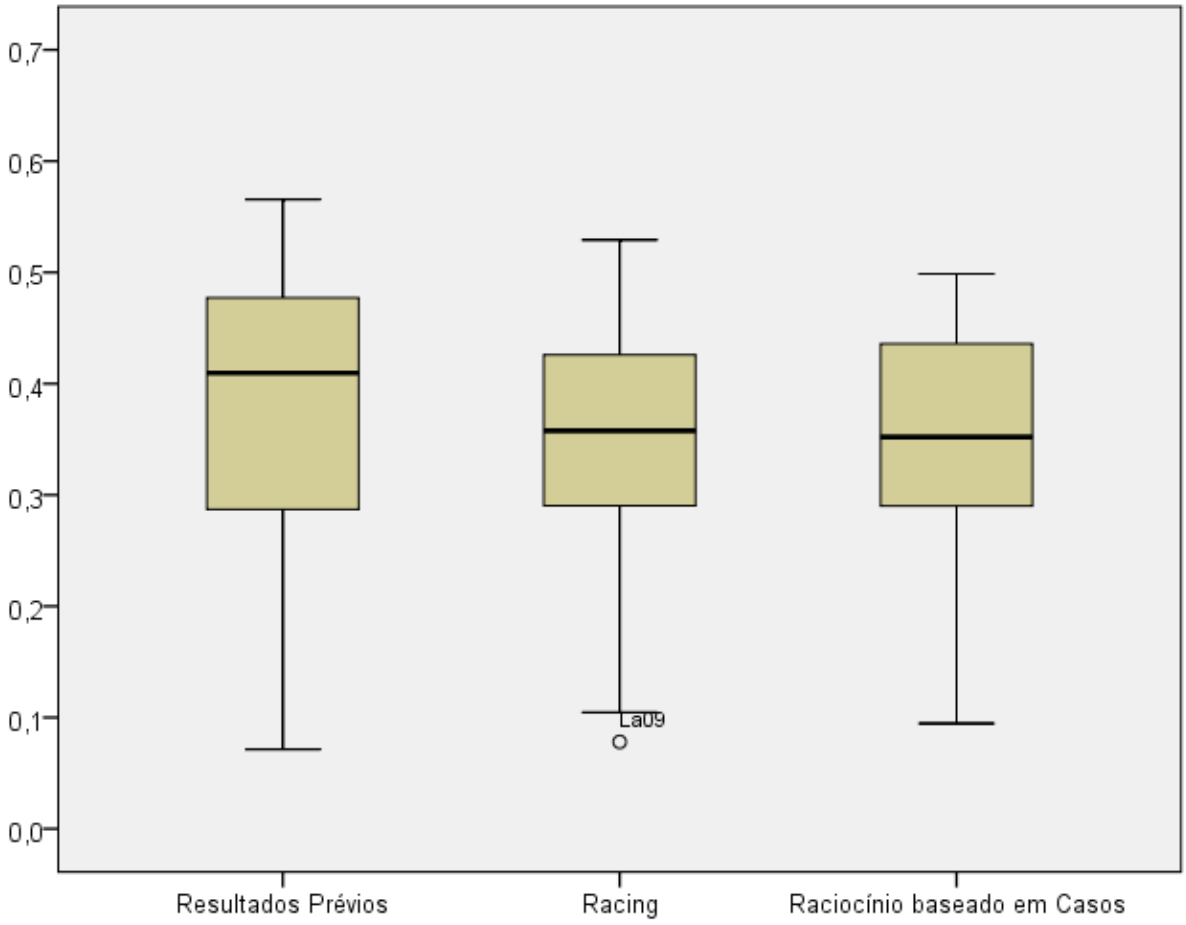

Figura 9.6 – Comparação do quociente dos valores médios da execução do sistema, entre os resultados prévios, os resultados de *Racing*, e os resultados de Raciocínio baseado em Casos

Tabela 9.23 – Resultado do teste *t* de Student para amostras emparelhadas: Resultados prévios *vs.* Raciocínio baseado em Casos e *Racing vs.* Raciocínio baseado em Casos

	Média da diferença	Desvio padrão da diferença	*t*	Graus de liberdade	*p_value*
Prévios *vs.* Raciocínio baseado em Casos	0,04042	0,05587	3,963	29	0,000
***Racing vs.* Raciocínio baseado em Casos**	0,00860	0,04240	1,111	29	0,276

Por outro lado, comparando os resultados do módulo de Raciocínio baseado em Casos com os resultados obtidos pelo módulo de *Racing* (Figura 9.6 e Tabela 9.22), não se verificam diferenças significativas nem na média nem no desvio padrão. Analisando a significância estatística destes resultados (Tabela 9.23), através da observação dos valores

$t(29) = 1{,}111; p > 0{,}05$, pode-se verificar que não existem diferenças estatisticamente significativas entre os resultados da abordagem de Raciocínio baseado em Casos face aos da abordagem baseada em *Racing*.

Para finalizar este estudo, os tempos de execução médios são apresentados na Tabela 9.24, onde é possível analisar que o módulo de Raciocínio baseado em Casos selecionou Meta-heurísticas com parâmetros considerados bastante eficientes, tendo em conta os obtidos nos resultados prévios, principalmente (Figura 9.7).

Tabela 9.24 – Tempos de execução médios do sistema com o módulo de Raciocínio baseado em Casos (em segundos)

Número de tarefas	Raciocínio baseado em Casos
10	1,02
15	1,32
20	1,84
30	6,32
50	27,03

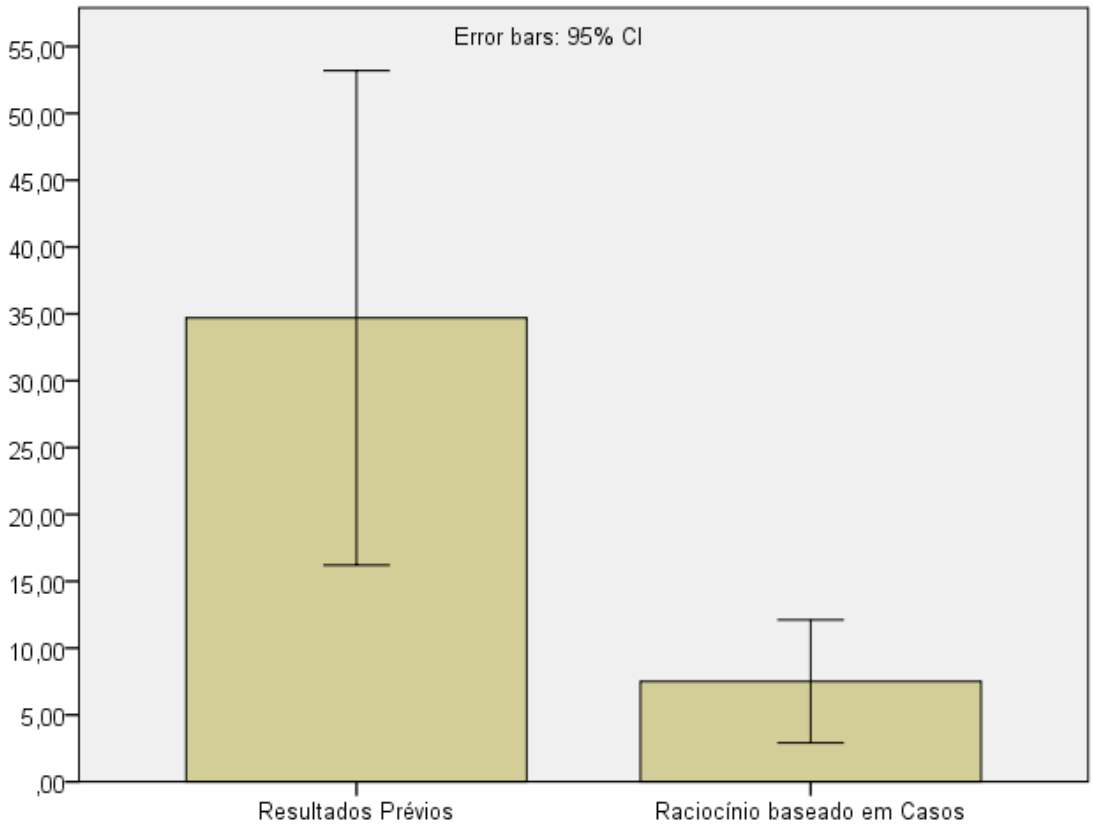

Figura 9.7 – Comparação dos tempos de execução médios, entre os resultados prévios e os resultados de Raciocínio baseado em Casos

9.3.4. Abordagem de *Racing* + Raciocínio baseado em Casos

De modo a tirar partido das potencialidades do módulo de *Racing* e do módulo de Raciocínio baseado em Casos, foi proposta uma abordagem híbrida, onde se emprega o *Racing* para inicializar a base de Casos e se utiliza o Raciocínio baseado em Casos para resolver novos casos com base na experiência passada.

A base de casos é inicializada com base nos parâmetros obtidos pelo estudo de *Racing*. Estes parâmetros estão apresentados na Tabela 9.25, Tabela 9.26, Tabela 9.27, Tabela 9.28, Tabela 9.29, e Tabela 9.30. É possível reparar em algumas diferenças face aos parâmetros dos resultados prévios (que deram origem aos casos iniciais do módulo de Raciocínio baseado em Casos). Por exemplo, o tamanho da lista tabu igual a 3 para instâncias com 10 tarefas ou a taxa de cruzamento dos Algoritmos Genéticos igual a 65% para instâncias de 15 tarefas.

Tabela 9.25 – Valores para os parâmetros da Pesquisa Tabu, no estudo de *Racing* + Raciocínio baseado em Casos

MH	Número de tarefas	NeighGen	SubNeigh	TabuList Len	StopCrit
TS	10	15%	100%	3	50
	15			1	
	20			2	125
	30			3	175
	50			5	200

Tabela 9.26 – Valores para os parâmetros dos Algoritmos Genéticos, no estudo de *Racing* + Raciocínio baseado em Casos

MH	Número de tarefas	InitPop Gen	PopSize	NumGen	CrossRate	MutRate
GA	10	15%	100%	100	75%	1%
	15			125	65%	
	20			150	75%	
	30			200		
	50		35%	400		

Tabela 9.27 – Valores para os parâmetros de *Simulated Annealing*, no estudo de *Racing* + Raciocínio baseado em Casos

MH	Número de tarefas	InitTemp	Alpha	Num ItK	StopCrit
SA	10	15	80%	5	30
	15			10	70
	20			20	
	30			30	90
	50				200

Tabela 9.28 – Valores para os parâmetros de Otimização por Colónia de Formigas, no estudo de *Racing* + Raciocínio baseado em Casos

MH	Número de tarefas	NumCol	Num Ants	Evap Rate	Alpha	Beta	StopCrit
ACO	10	1	20	80%	2	2	75
	15		50			1	100
	20		30		1	2	
	30		50		2		250
	50		75			1	350

Tabela 9.29 – Valores para os parâmetros de *Particle Swarm Optimization*, no estudo de *Racing* + Raciocínio baseado em Casos

MH	Número de tarefas	Num Part	Num It	Low Limit	Up Limit	Min Inertia	Max Inertia	C1	C2	Min Veloc	Max Veloc
PSO	10	30	1000	0	4	40%	95%	2	2	-4	4
	15	25	1500								
	20	75	1750								
	30		3500								
	50		4500								

Tabela 9.30 – Valores para os parâmetros de Colónia de Abelhas Artificiais, no estudo de *Racing* + Raciocínio baseado em Casos

MH	Número de tarefas	PopSize Sn	MaxFail	NumCycles
ABC	10	75	1250	2000
	15		1750	3000
	20	125	2000	3500
	30		2250	4000
	50			4500

De forma similar ao módulo de Raciocínio baseado em Casos, nesta abordagem híbrida são também inseridos 150 casos iniciais, que correspondem aos valores obtidos pelos parâmetros nas instâncias ao longo do estudo de *Racing* (resultados de 6 Meta-heurísticas em 25 instâncias).

Usando a base de casos inicial preenchida com base no estudo de *Racing*, procedeu-se depois à evolução do estudo com o uso de Raciocínio baseado em Casos, executando o sistema nas instâncias de teste.

Na Tabela 9.31 é apresentada a análise do quociente dos valores médios de *Cmax* do sistema *AutoDynAgents* de todas as abordagens, incluindo o *Racing* + Raciocínio baseado em Casos. Da análise do gráfico da Figura 9.8, é possível concluir quanto à vantagem da abordagem híbrida analisando a mediana e dispersão dos dados.

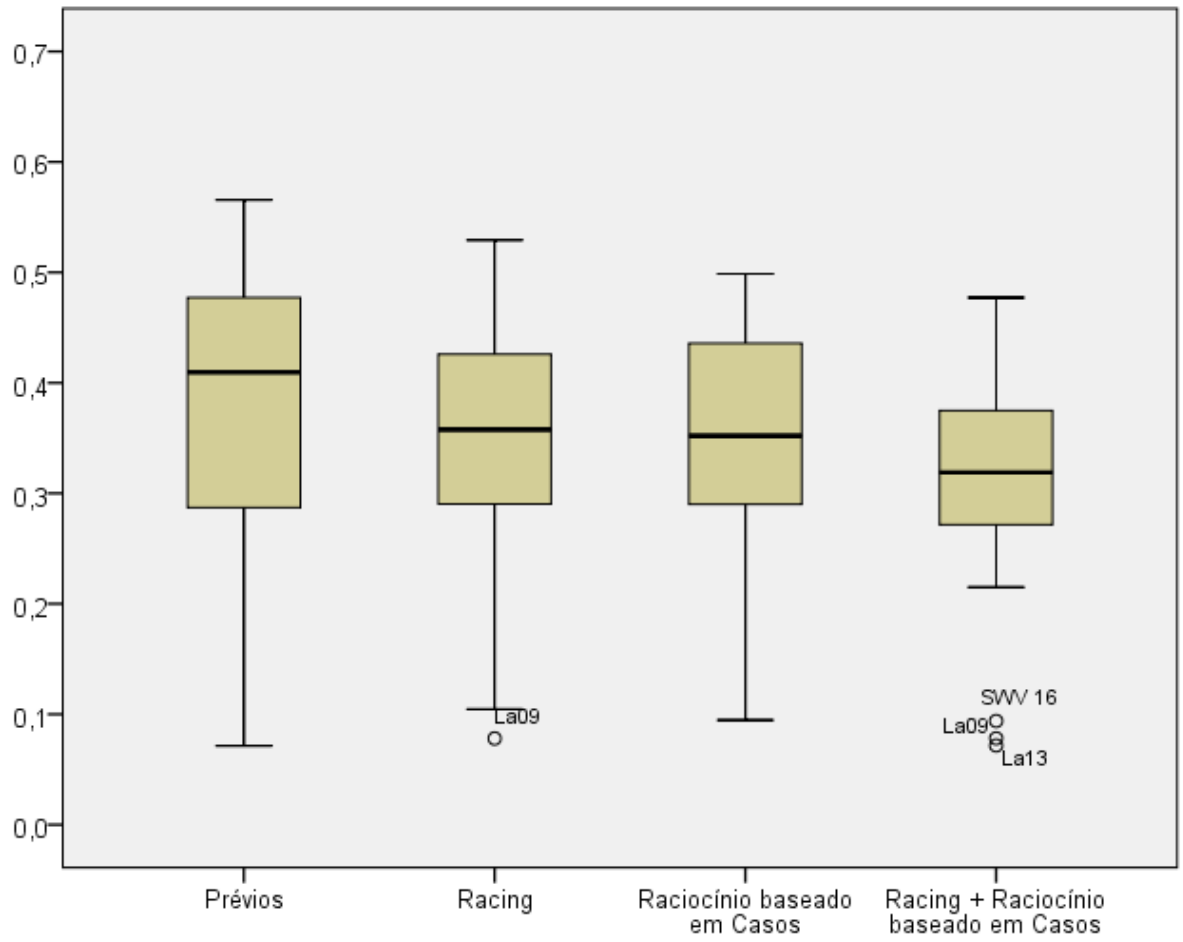

Figura 9.8 – Comparação do quociente dos valores médios da execução do sistema, entre os resultados prévios, os resultados de *Racing*, os resultados de Raciocínio baseado em Casos, e os resultados da abordagem híbrida *Racing* + Raciocínio baseado em Casos

Tabela 9.31 – Análise descritiva do quociente dos valores médios da execução do sistema após o estudo de *Racing* + Raciocínio baseado em Casos

	Valor mínimo	Valor máximo	Média	Desvio padrão
Resultados prévios	0,07	0,57	0,3792	0,13217
Racing	0,08	0,53	0,3474	0,11834
Raciocínio baseado em Casos	0,09	0,50	0,3388	0,10736
***Racing* + Raciocínio baseado em Casos**	0,07	0,48	0,3142	0,10999

Comparando os resultados da abordagem híbrida com os resultados do módulo de *Racing*, é possível verificar uma ligeira melhoria dos resultados, principalmente na dispersão, o que permite concluir que a inclusão do módulo de Raciocínio baseado em Casos consegue manter uma maior consistência dos resultados. Analisando a Tabela 9.31, nota-se uma média e um desvio padrão inferiores em relação ao módulo de *Racing*, e além disso um valor mínimo e um valor máximo inferiores, o que é importante, uma vez que se trata de um problema de minimização.

Na comparação da abordagem híbrida com os resultados do módulo de Raciocínio baseado em Casos, a melhoria já não é tão significativa, o que antecipa que a junção dos dois módulos poderá trazer vantagens. Analisando a Tabela 9.31, notam-se melhorias no valor mínimo e máximo, bem como na média, mas o desvio padrão é ligeiramente superior, embora a diferença seja pouco significativa.

Finalmente, são comparados os resultados da abordagem de *Racing* + Raciocínio baseado em Casos com os resultados prévios. E aqui evidenciam-se claras melhorias, o que permite concluir que há vantagens estatisticamente significativas na utilização de algoritmos de aprendizagem na afinação dos parâmetros de Meta-heurísticas em problemas de otimização. Analisando a Tabela 9.31 é possível reparar em melhorias expressivas, principalmente na média dos resultados e no desvio padrão. O valor mínimo foi igual mas o valor máximo bastante inferior. Finalmente, comparando as duas primeiras abordagens com a abordagem híbrida, é possível verificar quanto à vantagem de utilização desta última.

Neste ponto torna-se importante analisar a significância estatística destes resultados. Analisando a Tabela 9.32, na comparação entre os resultados prévios e os resultados da

abordagem híbrida, observando os valores $t(29) = 5{,}475; p < 0{,}05$, pode-se afirmar, com um grau de confiança de 95%, que existem diferenças estatisticamente significativas entre os resultados obtidos inicialmente (sem aprendizagem) e os resultados obtidos pela abordagem híbrida, permitindo concluir quanto à vantagem da utilização desta. Comparando os resultados da abordagem híbrida, com os resultados de *Racing*, e observando os valores $t(29) = 5{,}068; p < 0{,}05$, também se pode afirmar, com um grau de confiança de 95%, que existem diferenças estatisticamente significativas entre as duas abordagens, com vantagens para a abordagem híbrida. Finalmente, comparando os resultados de Raciocínio baseado em Casos com os resultados da abordagem híbrida, também é possível concluir, com um grau de confiança de 95%, que existem diferenças estatisticamente significativas entre as duas abordagens, pois $t(29) = 2{,}581; p < 0{,}05$.

É importante salientar que, conforme referido no início do capítulo, foram analisados os valores médios num total de 5 execuções por instância. De modo a completar o estudo, é possível comparar também os melhores resultados obtidos, para verificar se, além de melhorar os resultados médios, a abordagem híbrida também melhora os melhores resultados obtidos, face aos resultados prévios. Estes resultados são ilustrados na Figura 9.9. Da análise dos resultados, é possível verificar que a abordagem híbrida *Racing* + Raciocínio baseado em Casos obteve resultados melhores em 17 instâncias e obteve resultados iguais em 2 instâncias, o que permite concluir que os resultados em 19 instâncias foram melhores ou iguais do que os resultados prévios, o que corresponde a uma melhoria de 63,33%.

Tabela 9.32 – Resultado do teste t de Student para amostras emparelhadas: Resultados prévios vs. *Racing* + Raciocínio baseado em Casos, *Racing* vs. *Racing* + Raciocínio baseado em Casos, Raciocínio baseado em Casos vs. *Racing* + Raciocínio baseado em Casos

	Média da diferença	**Desvio padrão da diferença**	***t***	**Graus de liberdade**	***p_value***
Prévios *vs. Racing* + Raciocínio baseado em Casos	0,06496	0,06499	5,475	29	0,000
***Racing vs. Racing* + Raciocínio baseado em Casos**	0,03314	0,03582	5,068	29	0,000
Raciocínio baseado em Casos *vs. Racing* + Raciocínio baseado em Casos	0,02454	0,05208	2,581	29	0,015

Figura 9.9 – Comparação do quociente dos melhores valores obtidos da execução do sistema, entre os resultados prévios e os resultados de *Racing* + Raciocínio baseado em Casos

Por último, e apenas para completar o estudo, apresentam-se na Tabela 9.33 os tempos de execução médios obtidos pelo sistema *AutoDynAgents* com a inclusão da abordagem híbrida *Racing* + Raciocínio baseado em Casos. Comparando os tempos de execução médios entre todas as abordagens (Figura 9.10), é possível evidenciar que a abordagem híbrida melhorou significativamente em relação os resultados prévios, tendo havido também uma ligeira melhoria em relação à abordagem baseada de *Racing*. Não se evidenciam, no entanto, melhorias em termos de eficiência em relação à abordagem de Raciocínio baseado em Casos. Assim, além de melhorar a eficácia dos resultados, a abordagem híbrida permitiu melhorar também a eficiência do sistema, principalmente quando comparando com os resultados prévios (sem aprendizagem).

Tabela 9.33 – Tempos de execução médios do sistema com a abordagem híbrida de *Racing* + Raciocínio baseado em Casos (em segundos)

Número de tarefas	*Racing* + Raciocínio baseado em Casos
10	0,58
15	0,72
20	1,32
30	16,91
50	17,41

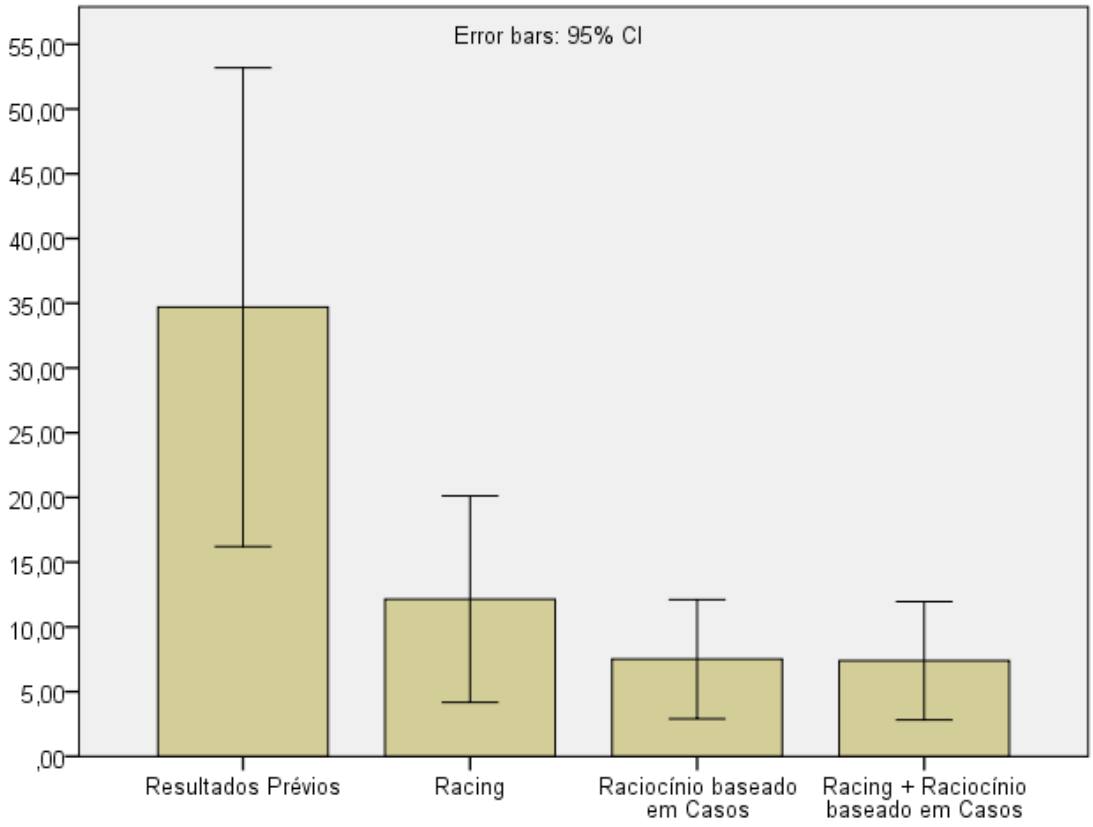

Figura 9.10 – Comparação dos tempos de execução médios, entre os resultados prévios, os resultados de *Racing*, os resultados de Raciocínio baseado em Casos, e os resultados da abordagem híbrida *Racing* + Raciocínio baseado em Casos

9.4. Sumário

Neste capítulo, foi apresentado o estudo computacional efetuado para a validação e análise do desempenho do sistema *AutoDynAgents,* na minimização de *Cmax*, com a incorporação do módulo de Auto-Otimização, proposto e desenvolvido no âmbito deste trabalho de doutoramento.

O estudo computacional foi dividido em duas partes principais, sendo que, na primeira foi efetuado um estudo do desempenho encontrado pela Aprendizagem para a Equipa e pela Aprendizagem Concorrente, de modo a suportar a decisão de qual das duas abordagens seria usada na implementação dos mecanismos de aprendizagem propostos, para a autoparametrização das Meta-heurísticas.

A segunda parte do estudo computacional teve como objetivo validar o desempenho do sistema com a incorporação dos módulos de aprendizagem no sistema *AutoDynAgents*. Primeiro, foram apresentados os resultados computacionais obtidos previamente, sem incorporação dos mecanismos de aprendizagem. Seguidamente apresentaram-se os resultados obtidos após a inclusão de cada um dos módulos de aprendizagem, *Racing* e Raciocínio baseado em Casos, separadamente. Finalmente, foram discutidos os resultados da abordagem híbrida *Racing* + Raciocínio baseado em Casos. Todos os resultados foram validados pela análise da significância estatística.

Foi possível verificar vantagem estatisticamente significativa na utilização do módulo de Auto-Otimização, independentemente da abordagem a usar. Todas as abordagens melhoraram os resultados obtidos pelo sistema *AutoDynAgents* em relação aos resultados prévios. No entanto, a abordagem híbrida *Racing* + Raciocínio baseado em Casos obteve melhores resultados médios. Além disso, foi possível melhorar também 63,33% dos melhores resultados obtidos, comparativamente aos resultados prévios. Dos resultados obtidos, foi possível concluir quanto à existência de evidência estatística na inclusão de aprendizagem no módulo de Auto-Otimização, no desempenho do sistema *AutoDynAgents*, quer em eficácia, o objetivo principal deste estudo, quer ainda em eficiência.

Capítulo 10. Conclusão

10.1. Introdução

A principal motivação deste trabalho surgiu da necessidade no desenvolvimento de abordagens que sejam capazes de controlar, coordenar e otimizar de forma adaptativa a resolução dos diferentes desafios existentes no problema de Escalonamento em ambientes reais de produção.

Este trabalho de doutoramento foi desenvolvido no âmbito do projeto de I&D *AutoDynAgents – Autonomic Agents with Self-Managing Capabilities for Dynamic Scheduling Support in a Cooperative Manufacturing System* (POCTI/EME-GIN/66848/2006), aprovado pela Fundação para a Ciência e a Tecnologia. O sistema *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.

A afinação de parâmetros nas Meta-heurísticas pode permitir uma maior flexibilidade e robustez mas requer uma inicialização cuidadosa. Os parâmetros podem ter uma grande influência na eficiência e eficácia do processo de pesquisa. Não se torna óbvia a definição *a priori* dos parâmetros a usar. Os valores para os parâmetros dependem do problema, das instâncias a tratar e do tempo disponível para a resolução do problema. Não existem valores "universais" para os parâmetros considerados para as diferentes Meta-heurísticas, havendo uma opinião generalizada que a sua definição deve resultar de um cuidadoso esforço experimental no sentido da respetiva afinação.

Assim, surgiu a necessidade de implementação de um módulo de Auto-Otimização para a seleção de uma determinada Meta-heurística, e especificação automática dos respetivos parâmetros, na resolução de novos problemas de escalonamento, numa perspetiva de suporte à decisão. Este módulo incorporou técnicas de aprendizagem, de modo a dotar o sistema da capacidade de aprender com a experiência adquirida na resolução de problemas anteriores similares.

Neste trabalho de doutoramento, pretendeu-se investigar o desenvolvimento de sistemas inteligentes para Escalonamento assistidos por aprendizagem, com recurso a aprendizagem por acumulação e interpretação da experiência. No âmbito deste trabalho foram integradas várias áreas de investigação desde a Aprendizagem Automática, Computação Autónoma, Meta-Heurísticas, até à Coordenação e Aprendizagem em Sistemas Multiagente.

10.2. Principais conclusões

De modo a dar uma contribuição para a resolução do problema de autoparametrização de Meta-heurísticas, na resolução do problema de escalonamento de tarefas de produção, foi proposto e desenvolvido um agente incorporando ideias da Computação Autónoma, designado de Auto-Otimização. Este agente é capaz de monitorizar o sistema *AutoDynAgents* e configurar os parâmetros das diferentes Meta-heurísticas, considerando as características de cada problema que surge no sistema.

O agente de Auto-Otimização proposto incorpora um mecanismo de aprendizagem baseada em *Racing* e outro em Raciocínio baseado em Casos, para implementar a capacidade de autoparametrização. Foi também proposta uma estratégia de aprendizagem híbrida, no sentido de tirar partido das potencialidades das duas técnicas de aprendizagem referidas.

O *Racing* permite que seja efetuado um estudo entre várias combinações de parâmetros, para que seja possível determinar a melhor combinação na aplicação de uma determinada Meta-heurística. O Raciocínio baseado em Casos dá ao sistema a capacidade de evoluir e aprender com a experiência, uma vez que se baseia em casos passados e retém toda a informação da experiência para uso futuro. Na abordagem híbrida, o *Racing* é utilizado para especificar uma parametrização inicial ao Raciocínio baseado em Casos, que permite ao sistema evoluir e aprender através da experiência.

De modo a validar a vantagem de utilização do agente de Auto-Otimização proposto, foi realizado um estudo computacional, dividido em duas partes principais. Na primeira parte foi efetuado um estudo do desempenho de duas estratégias de aprendizagem distintas: Aprendizagem para a Equipa e Aprendizagem Concorrente. O objetivo deste estudo consistiu em suportar a decisão de qual das duas abordagens multiagente seria usada na implementação dos mecanismos de aprendizagem propostos, para a autoparametrização das Meta-heurísticas. Com este estudo, foi possível evidenciar vantagem na utilização da

abordagem de Aprendizagem para a Equipa, o que se poderá justificar pelo facto do agente de Auto-Otimização possuir uma visão global de todo o sistema, tentando otimizar o plano de escalonamento como um todo. Na abordagem de Aprendizagem Concorrente, os vários agentes aprendizes são egoístas e competitivos entre si, de modo a tentar melhorar o seu plano local, não se preocupando com o plano global.

A segunda parte do estudo computacional teve como objetivo validar o desempenho do sistema com a incorporação dos módulos de aprendizagem no sistema *AutoDynAgents*. Foi possível verificar vantagem estatisticamente significativa na utilização do módulo de Auto-Otimização, independentemente da abordagem a usar. Todas as abordagens melhoraram os resultados obtidos pelo sistema *AutoDynAgents* em relação aos resultados obtidos previamente sem aprendizagem. No entanto, a abordagem híbrida (*Racing* + Raciocínio baseado em Casos) evidenciou-se como a mais eficaz e eficiente. Além disso, foi possível melhorar também 63,33% dos melhores resultados obtidos, comparativamente aos resultados prévios. Foi possível concluir quanto à existência de evidência estatística da contribuição de aprendizagem no módulo de Auto-Otimização, no desempenho do sistema *AutoDynAgents*, quer em eficácia, o objetivo principal deste estudo, quer ainda em eficiência.

Assim sendo, foi possível concluir acerca da vantagem na inclusão de aprendizagem no processo de autoparametrização de Meta-heurísticas, aplicadas num Sistema Multiagente, para a resolução do problema de escalonamento de tarefas em sistemas de produção.

10.3. Contribuições

Identifica-se como principal contribuição deste trabalho a definição de uma plataforma baseada em Sistemas Multiagente e em técnicas inspiradas em sistemas biológicos com capacidades de aprendizagem e autogestão para a resolução de problemas de escalonamento de tarefas em sistemas de produção. O sistema *AutoDynAgents* foi equipado com capacidade de diagnóstico e resolução de problemas, incorporando os conceitos da Computação Autónoma. Foi também dotado da capacidade de aprendizagem para autonomamente definir os parâmetros das Meta-heurísticas a usar, através de um comportamento de Auto-Otimização.

Ao longo deste trabalho, foi efetuado o levantamento e revisão do estado da arte e descrição de:

- Abordagens de resolução de problemas de Otimização Combinatória, onde se inclui o problema de escalonamento de tarefas em sistemas de produção;
- Meta-heurísticas utilizadas ao longo deste trabalho, com foco em Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, Otimização por Colónia de Formigas, *Particle Swarm Optimization*, e Colónias de Abelhas Artificiais;
- Abordagens para a afinação de parâmetros de Meta-heurísticas;
- Características, modelos e coordenação de Sistemas Multiagente;
- Comportamentos de autogestão e desafios da Computação Autónoma;
- Aprendizagem em Sistemas Multiagente, com especial foco em Aprendizagem para a Equipa e Aprendizagem Concorrente;
- Aprendizagem para afinação de parâmetros, com ênfase nos métodos utilizados neste trabalho: *Racing* e Raciocínio baseado em Casos.

Outras contribuições incluem:

- Modelação, especificação e implementação de mecanismos de aprendizagem para dotar o sistema *AutoDynAgents* com a capacidade de autoparametrização. A proposta das abordagens de aprendizagem é assente em *Racing* e Raciocínio baseado em Casos. É também proposta uma abordagem híbrida que tira partido das vantagens dos mecanismos;
- Com base nos algoritmos de aprendizagem referidos, foi proposto e especificado um módulo de Auto-Otimização para integração no sistema *AutoDynagents*, de modo que este pudesse resolver novas instâncias do problema de Escalonamento com o mínimo de intervenção humana;
- Realização de um estudo computacional entre abordagens de Aprendizagem para a Equipa e Aprendizagem Concorrente, para afinação de parâmetros no Sistema Multiagente *AutoDynAgents*. Este estudo permitiu concluir e suportar a decisão acerca de qual a abordagem a seguir;
- Realização de um estudo computacional entre as várias abordagens de aprendizagem propostas, que permitiu concluir acerca da vantagem da sua utilização.

Identifica-se também como contribuições, no âmbito desta tese de doutoramento, a escrita de artigos científicos e apresentação do trabalho em conferências internacionais. As contribuições científicas associadas a este trabalho de doutoramento são:

- 1 publicação em revista internacional ISI (IF: 2.679)
    - Pereira, I. e Madureira, A. (2013), Self-Optimization module for Scheduling using Case-based Reasoning, Applied Soft Computing 13(3), 1419-1432
- 3 publicações em capítulos de livros
    - Madureira, A. e Pereira, I. (2010), Self-Optimization for Dynamic Scheduling in Manufacturing Systems, Technological Developments in Networking, Education and Automation, pp. 421-426, Springer Netherlands
    - Pereira, I., Madureira, A., e Oliveira, P. (2013). Meta-heuristics Self-Parameterization in a Multi-agent Scheduling System Using Case-Based Reasoning, Computational Intelligence and Decision Making (61), pp. 99-109, Springer Netherlands
    - Pereira, I., Madureira, A., Oliveira, P., e Abraham, A. (2013), Tuning Meta-heuristics Using Multi-agent Learning in a Scheduling System. Transactions on Computational Science, Springer Verlag
- 7 publicações em conferências internacionais
    - Pereira, I. e Madureira, A. (2010), Case-based Reasoning for Self-Optimizing Behavior, 2010 IEEE International Conference on Systems Man and Cybernetics (SMC)
    - Pereira, I. e Madureira, A. (2010), Meta-heuristics Tunning using CBR for Dynamic Scheduling, 2010 IEEE 9th International Conference on Cybernetic Intelligent Systems (CIS)
    - Pereira, I. e Madureira, A. (2010), Auto-parametrização de Meta-heurísticas para Escalonamento Dinâmico. WACI'10 - 5th Workshop on Applications of Computational Intelligence
    - Pereira, I., e Madureira, A. (2010). Self-optimizing through CBR learning. 2010 IEEE Congress on Evolutionary Computation (CEC)
    - Pereira, I. e Madureira, A. (2010). Self-Optimization Aspects for Dynamic Scheduling, 24th European Conference on Operational Research (EURO XXIV)

    - Pereira, I., Madureira, A., e Oliveira, P. (2011), Case-based Reasoning for Meta-heuristics Self-Parameterization in a Multi-Agent Scheduling System, 2nd International Symposium on Computational Intelligence for Engineering Systems (ISCIES' 2011)
    - Pereira, I., Madureira, A., e Oliveira, P. (2012), Multi-apprentice learning for Meta-heuristics Parameter Tuning in a Multi Agent Scheduling System. 2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC)
- 2 co-orientações de alunos;
    - Ricardo Bártolo (2013), Tutor de Meta-heurísticas, Relatório final de curso da Licenciatura em Engenharia Informática, Instituto Superior de Engenharia do Porto
    - Diamantino Falcão (2014), Híper-heurísticas com Aprendizagem, Tese de Mestrado em Engenharia Informática, Ramo de Tecnologias do Conhecimento e Decisão, Instituto Superior de Engenharia do Porto
- Participação em 2 projetos de investigação financiados pela Fundação para a Ciência e Tecnologia.
    - Autonomic Agents with Self-Managing Capabilities for Dynamic Scheduling Support in a Cooperative Manufacturing System (AutoDynAgents), PTDC/EME-GIN/66848/2006, 2007 - 2010
    - Adaptive Decision Support System for Interactive Scheduling with MetaCognition and User Modelling Experience (ADSyS), PTDC/EME-GIN/109956/2009, 2010-2014

Refira-se ainda a participação como co-autor em 1 publicação em revista internacional, 6 publicações em capítulos de livros, e 17 publicações em conferências internacionais.

Este trabalho de doutoramento serviu também de inspiração para outros trabalhos. Destaca-se a utilização de Raciocínio baseado em Casos na integração de novas tarefas num ambiente de escalonamento dinâmico (Madureira *et al.*, 2013b) e também a utilização da mesma técnica para aprendizagem adaptativa numa *framework* baseada em agentes de simulação de mercados de eletricidade (Pinto, 2011).

10.4. Limitações e perspetivas de trabalho futuro

Na realização deste trabalho foram detetadas algumas limitações e vulnerabilidades subjacentes ao desenvolvimento e implementação do módulo de Auto-Otimização. Uma dessas limitações é o facto de apenas se ter considerado a heurística *SeqNivel* para a geração da solução inicial das Meta-heurísticas. A escolha desta heurística deveu-se ao facto da mesma ter obtido anteriormente bons resultados comparativamente a outras. No entanto, considera-se importante analisar o comportamento e evolução do sistema com outras heurísticas de geração da solução inicial, de modo a ser possível verificar se a heurística *SeqNivel* é ou não a mais adequada, dependendo da evolução das parametrizações das Meta-heurísticas na resolução de problemas de escalonamento diferentes.

Outro aspeto limitativo relaciona-se com o crescimento da base de casos. Ao longo do tempo, a base de casos cresce exponencialmente com a resolução de novos casos, podendo levar a uma degradação do desempenho do módulo de Auto-Otimização. Uma vez que existem casos que vão ficando obsoletos, considera-se importante a remoção destes, através do uso de técnicas de agrupamento de dados de modo a detetar grupos de casos similares, para ser possível eliminar os piores casos, uma vez que se considera que estes têm uma menor probabilidade de serem usados no futuro, e também uma menor probabilidade de levar a melhores resultados.

Saliente-se ainda o facto de não se garantir a constante evolução dos resultados, uma vez que tanto o módulo de Auto-Otimização como as Meta-heurísticas em si têm intrínseca alguma aleatoriedade, que poderá influenciar o respetivo desempenho.

De modo a superar as limitações identificadas, sugere-se como trabalho futuro, a integração de outras heurísticas de geração de solução inicial, além da *SeqNivel*, e também na referida manutenção da Base de Casos, de forma a limpar casos redundantes, através do uso de técnicas de agrupamento de dados.

Em relação ao futuro, tal como Eiben e Smit (2012) referem, prevê-se uma mudança na atitude a larga escala no uso de afinadores de parâmetros, principalmente em publicações científicas e aplicações, por parte de investigadores e na indústria que utilize algoritmos de pesquisa.

Considera-se importante, no futuro, focar a investigação em parametrização *online* de modo a ultrapassar as desvantagens das abordagens de parametrização *offline*, usadas

neste trabalho. As abordagens de parametrização *online* monitorizam o processo de pesquisa e ajustam os valores dos parâmetros em tempo de execução. A capacidade de dotar os algoritmos de pesquisa com comportamentos de adaptação autónoma de parâmetros tem gerado um interesse crescente dentro da comunidade de investigação.

A área das Híper-heurísticas é igualmente muito promissora, onde uma abordagem de alto nível é utilizada para escolher e aplicar uma heurística de baixo-nível apropriada para instância do problema em questão, automatizando assim o processo de escolha de algoritmos de pesquisa. Embora já esteja a ser desenvolvido algum trabalho nesta área, através de um aluno de Mestrado, com a incorporação de técnicas de aprendizagem (especificamente *Q-learning*), considera-se importante investir na investigação e no desenvolvimento de Híper-heurísticas que, além de selecionar técnicas de pesquisa, possam afinar os parâmetros das mesmas.

Tendo em conta estes últimos aspetos, e observando os promissores resultados obtidos, considera-se interessante o desenvolvimento de uma *framework* baseada em *Racing* e Raciocínio baseado em Casos, aplicável tanto a parametrização *offline* como *online*, de modo a ser possível automatizar o processo de seleção e afinação de Meta-heurísticas, com o mínimo de intervenção humana. Esta *framework* poderá estar preparada para lidar com os mais variados tipos de problemas de otimização, além do problema de escalonamento.

Considera-se também importante a proposta e implementação de outras técnicas, para se contrastar com as abordadas ao longo deste trabalho, como, por exemplo, Redes Neuronais, para efetuar a previsão da Meta-heurística a utilizar, e Lógica Difusa (*Fuzzy Logic*), para introduzir abstração aos valores dos parâmetros das Meta-heurísticas.

Para terminar, e uma vez que a validação deste trabalho se baseou em problemas de *benchmark* encontrados na literatura, considera-se de elevada importância conduzir a investigação para problemas de escalonamento dinâmico, encontrados na realidade industrial. É importante sublinhar, no entanto, que este trabalho está preparado para lidar com dinamismo em problemas de escalonamento de tarefas em sistemas de produção.

Bibliografia

Aamodt, A., e Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications, 7*(1), 39-59.

Adams, J., Balas, E., e Zawack, D. (1988). The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science, 34*(3).

Adenso-Diaz, B., e Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research, 54*(1), 99-114.

Alonso, E., D'inverno, M., Kudenko, D., Luck, M., e Noble, J. (2001). Learning in multi-agent systems. *The Knowledge Engineering Review, 16*(03), 277-284.

Alpaydin, E. (2004). *Introduction to machine learning*: MIT press.

Applegate, D. L., Bixby, R. E., Chvátal, V., e Cook, W. J. (2006). *The Traveling Salesman Problem*: Princeton University Press.

Applegate, D. L., e Cook, W. J. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on computing, 3*(2), 149-156.

Artiba, A., e Elmaghraby, S. E. (1997). *The planning and scheduling of production systems*: Chapman & Hall.

Bäck, T. (1994). *Evolutionary algorithms in theory and practice*.

Bäck, T. (2001). Introduction to the Special Issue: self-adaptation. *Evolutionary Computation, 9*(2).

Bahrami, M., Faraahi, A., e Rahmani, A. M. (2010). *AGC4ISR, New Software Architecture for Autonomic Grid Computing.* Paper presented at the International Conference on Intelligent Systems, Modelling and Simulation (ISMS).

Baiying, S., e Wei, H. (2008). *Modeling and Development of Multi-agent Traffic Control Experimental System Based on Petri Net.* Paper presented at the International Conference on Intelligent Computation Technology and Automation (ICICTA).

Baker, J. E. (1987). *Reducing bias and inefficiency in the selection algorithm*. Paper presented at the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, Cambridge, Massachusetts, USA.

Baker, K. (1974). *Introduction to sequencing and scheduling*. New York: Wiley.

Balaji, P. G., e Srinivasan, D. (2010). Multi-Agent System in Urban Traffic Signal Control. *IEEE Computational Intelligence Magazine, 5*(4), 43-51.

Balaprakash, P., Birattari, M., e Stützle, T. (2007). Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement *Hybrid Metaheuristics* (pp. 108-122): Springer.

Balaprakash, P., Birattari, M., Stützle, T., e Dorigo, M. (2009). Adaptive sample size and importance sampling in estimation-based local search for the probabilistic traveling salesman problem. *European Journal of Operational Research, 199*(1), 98-110.

Bangerth, W., Klie, H., Matossian, V., Parashar, M., e Wheeler, M. (2005). An Autonomic Reservoir Framework for the Stochastic Optimization of Well Placement. *Cluster Computing, 8*(4), 255-269.

Banharnsakun, A., Sirinaovakul, B., e Achalakul, T. (2012). Job Shop Scheduling with the Best-so-far ABC. *Engineering Applications of Artificial Intelligence, 25*(3), 583-593.

Bartz-Beielstein, T. (2006). *Experimental research in evolutionary computation*: Springer Berlin.

Bartz-Beielstein, T., e Markon, S. (2004). *Tuning search algorithms for real-world applications: A regression tree based approach.* Paper presented at the Congress on Evolutionary Computation (CEC).

Bartz-Beielstein, T., Parsopoulos, K., e Vrahatis, M. (2004). *Analysis of Particle Swarm Optimization Using Computational Statistics.* Paper presented at the International Conference Numerical Analysis and Applied Mathematics (ICNAAM).

Battiti, R., e Brunato, M. (2010). Reactive search optimization: learning while optimizing *Handbook of Metaheuristics* (pp. 543-571): Springer.

Battiti, R., Brunato, M., e Mascia, F. (2008). *Reactive search and intelligent optimization*: Springer Publishing Company, Incorporated.

Beasley, J. E. (1990). OR-Library: distributing test problems by electronic mail. *Journal of the operational research society*, 1069-1072.

Becker, S. (2004). *Racing-Verfahren für Tourenplanungsprobleme.* Diplomarbeit, Technische Universität Darmstadt, Darmstadt, Germany.

Beddoe, G., e Petrovic, S. (2006). Selecting and weighting features using a genetic algorithm in a case-based reasoning approach to personnel rostering. *European Journal of Operational Research, 175*(2), 649-671.

Beddoe, G., Petrovic, S., e Li, J. (2009). A hybrid metaheuristic case-based reasoning system for nurse rostering. *Journal of Scheduling, 12*(2), 99-119.

Bellifemine, F. L., Caire, G., e Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*: John Wiley & Sons.

Bernon, C., Cossentino, M., e Pavón, J. (2005). An Overview of Current Trends in European AOSE Research. *Informatica, 29*.

Besten, M. L. den. (2004). *Simples Metaheuristics for Scheduling: An empirical investigation into the application of iterated local search to deterministic scheduling problems with tardiness penalities.* (PhD thesis), Technische Universität, Darmstadt, Germany.

Bezek, A., Gams, M., e Bratko, I. (2006). *Multi-agent strategic modeling in a robotic soccer domain*. Paper presented at the Fifth international joint conference on Autonomous agents and multiagent systems, Hakodate, Japan.

Bigus, J. P., Schlosnagle, D. A., Pilgrim, J. R., Mills Iii, W. N., e Diao, Y. (2002). ABLE: A toolkit for building multiagent autonomic systems. *IBM Systems Journal, 41*(3), 350-371.

Birattari, M. (2009). *Tuning Metaheuristics: A Machine Learning Perspective*: Springer Publishing Company, Incorporated.

Birattari, M., Balaprakash, P., e Dorigo, M. (2007). The ACO/F-RACE algorithm for combinatorial optimization under uncertainty *Metaheuristics* (pp. 189-203): Springer.

Birattari, M., Stützle, T., Paquete, L., e Varrentrapp, K. (2002). *A racing algorithm for configuring metaheuristics.* Paper presented at the Genetic and evolutionary computation conference.

Bishop, C. M. (1995). *Neural networks for pattern recognition*: Oxford University Press.

Blazewicz, J., Ecker, K. H., Pesch, E., Smith, G., e Weglarz, J. (2001). *Scheduling Computer and Manufacturing Processes*. New York: Springer.

Blum, A., e Mitchell, T. (1998). *Combining labeled and unlabeled data with co-training.* Paper presented at the Eleventh annual conference on Computational learning theory.

Blum, C., e Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv., 35*(3), 268-308.

Bonhomme, C., Feltus, C., e Khadraoui, D. (2010). *A multi-agent based decision mechanism for incident reaction in telecommunication network*. Paper presented at the ACS/IEEE International Conference on Computer Systems and Applications - AICCSA 2010.

Bouabda, R., Jarboui, B., Eddaly, M., e Rebaï, A. (2011). A branch and bound enhanced genetic algorithm for scheduling a flowline manufacturing cell with sequence dependent family setup times. *Comput. Oper. Res., 38*(1), 387-393.

Box, G., Hunter, J., e Hunter, W. (2005). *Statistics for experimenters: design, innovation, and discovery* (Vol. 2): Wiley Online Library.

Box, G., Hunter, W., e Hunter, J. (1978). Statistics for experimenters. *John Willey, New York*.

Box, J. F. (1987). Guinness, Gosset, Fisher, and Small Samples. *Statistical Science, 2*, 45-52.

Breiman, L. (2001). Random forests. *Machine learning, 45*(1), 5-32.

Breiman, L., Friedman, J., Stone, C. J., e Olshen, R. A. (1984). *Classification and regression trees*: CRC press.

Brindle, A. (1981). *Genetic Algorithms for Function Optimization.* (PhD thesis), University of Alberta, Edmonton, Alberta, Canada.

Brunato, M., e Battiti, R. (2008). Rash: A self-adaptive random search method *Adaptive and Multilevel Metaheuristics* (pp. 95-117): Springer.

Brustolini, J. (1991). Autonomous Agents: characterization and requirements *Technical Report CMU-CS-91-204*. Pittsburgh, Pennsylvania, United States: Carnegie Mellon.

Burgard, W., Moors, M., Stachniss, C., e Schneider, F. E. (2005). Coordinated multi-robot exploration. *IEEE Transactions on Robotics, 21*(3), 376-386.

Burke, E. K., MacCarthy, B. L., Petrovic, S., e Qu, R. (2003). Knowledge discovery in a hyper-heuristic for course timetabling using case-based reasoning *Practice and Theory of Automated Timetabling IV* (pp. 276-287): Springer.

Burke, E. K., MacCarthy, B. L., Petrovic, S., e Qu, R. (2006a). Multiple-retrieval case-based reasoning for course timetabling problems. *Journal of the Operational Research Society, 57*(2), 148-162.

Burke, E. K., Petrovic, S., e Qu, R. (2006b). Case-based heuristic selection for timetabling problems. *Journal of Scheduling, 9*(2), 115-132.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications, 45*(1), 41-51.

Chakhlevitch, K., e Cowling, P. (2008). Hyperheuristics: recent developments *Adaptive and multilevel metaheuristics* (pp. 3-29): Springer.

Chan, H., Segal, A., Arnold, B., e Whalley, I. (2005). *How Can We Trust an Autonomic System to Make the Best Decision?* Paper presented at the Second International Conference on Automatic Computing.

Chen, J. R., Wolfe, S. R., e Wragg, S. D. (2000). *A distributed multi-agent system for collaborative information management and sharing*. Paper presented at the Ninth international conference on Information and knowledge management (CIKM '00), New York, USA.

Chen, P-H., e Shahandashti, S. M. (2009). Hybrid of genetic algorithm and simulated annealing for multiple project scheduling with multiple resource constraints. *Automation in Construction, 18*(4), 434-443.

Chiarandini, M. (2005). *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems.* (PhD thesis), Technische Universität, Darmstadt, Germany.

Chira, C., e Dumitrescu, D. (2006). *Development of a Multi-Agent Information Management System*. Paper presented at the 6th International Conference on Recent Advances in Soft Computing (RASC 2006), Canterbury, UK.

Chong, C. S., Low, M. Y. H., Sivakumar, A. I., e Gay, K. L. (2006). *A bee colony optimization algorithm to job shop scheduling.* Paper presented at the Winter Simulation Conference WSC 06.

Chopra, I., e Singh, M. (2011). SASM- An Approach towards Self-protection in Grid Computing. In S. Dua, S. Sahni e D. P. Goyal (Eds.), *Information Intelligence, Systems, Technology and Management* (Vol. 141, pp. 149-159): Springer Berlin Heidelberg.

Chou, F-D. (2012). Particle swarm optimization with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessor tasks. *International Journal of Production Economics*.

Clune, J., Goings, S., Punch, B., e Goodman, E. (2005). *Investigations in meta-GAs: panaceas or pipe dreams?* Paper presented at the Workshops on Genetic and evolutionary computation.

Coelho, H. (1995). *Inteligência artificial em 25 lições*: Fundação Calouste Gulbenkian.

Coello, J., e dos Santos, R. (1999). Integrating CBR and heuristic search for learning and reusing solutions in real-time task scheduling *Case-Based Reasoning Research and Development* (pp. 89-103): Springer.

Conover, W. J. (1999). *Practical nonparametric statistics*: John Wiley&Sons, New York.

Cook, J. E., e Tauritz, D. R. (2010). *An exploration into dynamic population sizing.* Paper presented at the 12th annual conference on Genetic and evolutionary computation.

Cook, S. A. (1971). *The complexity of theorem-proving procedures*. Paper presented at the Third annual ACM symposium on Theory of computing, Shaker Heights, Ohio, USA.

Cortes, C., e Vapnik, V. (1995). Support-vector networks. *Machine learning, 20*(3), 273-297.

Cotta, C., Sevaux, M., e Sörensen, K. (2008). *Adaptive and multilevel metaheuristics* (Vol. 136): Springer.

Coy, S., Golden, B., Runger, G., e Wasil, E. (2001). Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics, 7*(1), 77-97.

Darwin, C. (1859). *On the Origin of the Species by Means of Natural Selection: Or, The Preservation of Favoured Races in the Struggle for Life*: John Murray.

Das, R., Kephart, J. O., Lefurgy, C., Tesauro, G., Levine, D. W., e Chan, H. (2008). *Autonomic multi-agent management of power and performance in data centers*. Paper presented at the 7th international joint conference on Autonomous agents and multiagent systems: industrial track, Estoril, Portugal.

Dasgupta, D. (1998). *Artficial Immune Systems and Their Applications*: Springer-Verlag New York, Inc.

Davis, R. (1980). Report on the Workshop on Distributed Artificial Intelligence. *SIGART Newsletter*.

De Giovanni, L., e Pezzella, F. (2010). An Improved Genetic Algorithm for the Distributed and Flexible Job-shop Scheduling problem. *European Journal of Operational Research, 200*(2), 395-408.

De Jong, K. (1975). *Analysis of the behavior of a class of genetic adaptive systems.* (PhD thesis), University of Michigan, Ann Arbor, Michigan, United States.

De Jong, K. (2007). Parameter setting in EAs: a 30 year perspective *Parameter Setting in Evolutionary Algorithms* (pp. 1-18): Springer.

De Meo, P., Quattrone, G., e Ursino, D. (2011). Integration of the HL7 Standard in a Multiagent System to Support Personalized Access to e-Health Services. *IEEE Transactions on Knowledge and Data Engineering, 23*(8), 1244-1260.

Dean, A., e Voss, D. (1999). *Design and analysis of experiments*: Springer-Verlag New York.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering, 186*(2), 311-338.

Decker, K., e Sycara, K. (1997). Intelligent Adaptive Information Agents. *Journal of Intelligent Information Systems, 9*(3), 239-260.

Dempster, A., Laird, N., e Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1-38.

Desai, P., Loke, S. W., Desai, A., e Singh, J. (2011). *Multi-agent based vehicular congestion management.* Paper presented at the Intelligent Vehicles Symposium (IV), 2011 IEEE.

Dobslaw, F. (2010). *A parameter tuning framework for metaheuristics based on design of experiments and artificial neural networks.* Paper presented at the International Conference on Computer Mathematics and Natural Computing 2010.

Dorigo, M., Maniezzo, V., e Colorni, A. (1991). The Ant System: An Autocatalytic Optimizing Process *Technical report nº 91-016*.

Dorigo, M., e Stützle, T. (2003). The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In F. Glover e G. Kochenberger (Eds.), *Handbook of Metaheuristics* (Vol. 57, pp. 250-285): Springer US.

Draper, C., e Sweitzer, J. (2006). Architecture Overview for Autonomic Computing *Autonomic Computing* (pp. 71-98): CRC Press.

Dréo, J. (2009). *Using performance fronts for parameter setting of stochastic metaheuristics.* Paper presented at the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers.

Dunin-Keplicz, B. M., e Verbrugge, R. (2010). *Teamwork in Multi-Agent Systems: A Formal Approach*: Wiley Publishing.

Eiben, A. E., e Smit, S. K. (2012). Evolutionary algorithm parameters and methods to tune them *Autonomous Search* (pp. 15-36): Springer.

Epstein, S. L., Freuder, E. C., Wallace, R., Morozov, A., e Samuels, B. (2006). *The adaptive constraint engine.* Paper presented at the Principles and Practice of Constraint Programming-CP 2002.

Essafi, I., Mati, Y., e Dauzère-Pérès, S. (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research, 35*(8), 2599-2616.

Estlin, T., Gaines, D., Fisher, F., e Castano, R. (2005). *Coordinating multiple rovers with interdependent science objectives*. Paper presented at the Fourth international joint conference on Autonomous agents and multiagent systems, The Netherlands.

Fang-Chang, L., e Chien-Nan, K. (2002). *Cooperative multi-agent negotiation for electronic commerce based on mobile agents.* Paper presented at the IEEE International Conference on Systems, Man and Cybernetics, 2002

Femal, M. E., e Freeh, V. W. (2005). *Boosting Data Center Performance Through Non-Uniform Power Allocation*. Paper presented at the Second International Conference on Autonomic Computing ICAC 2005.

Feo, T., e Resende, M. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett., 8*(2), 67-71.

Fisher, H., e Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling, 3*, 225-251.

Fitzgerald, B. (2012). Software Crisis 2. 0. *Computer, 45*(4), 89-91.

Franklin, S., e Graesser, A. (1997). *Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents*. Paper presented at the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages.

French, S. (1982). *Sequencing and Scheduling: An introduction to the Mathematics of the Job-Shop*. Chichester: Ellis Horwood.

Gagliolo, M., e Schmidhuber, J. (2010). Algorithm selection as a bandit problem with unbounded losses *Learning and Intelligent Optimization* (pp. 82-96): Springer.

Ganek, A. (2006). Overview of Autonomic Computing *Autonomic Computing* (pp. 3-18): CRC Press.

Garland, A., e Alterman, R. (2004). Autonomous agents that learn to better coordinate. *Autonomous Agents and Multi-Agent Systems, 8*(3), 267-301.

Gâteau, B., Khadraoui, D., e Feltus, C. (2009). *Multi-agents system service based platform in telecommunication security incident reaction*. Paper presented at the Second international conference on Global Information Infrastructure Symposium, Hammamet, Tunisia.

Geem, Z. W. (2000). *Optimal design of water distribution networks using harmony search.* Korea University.

Gentner, D. (1983). Structure-Mapping: A Theoretical Framework for Analogy*. *Cognitive science, 7*(2), 155-170.

Gilbert, M. (2005). A theoretical framework for the understanding of teams. In N. Gold (Ed.), *Teamwork* (pp. 22–32). Basingstoke, UK: Palgrave McMillan.

Gleizes, M. P., Link-Pezet, J., e Glize, P. (2000). *An adaptive multi-agent tool for electronic commerce.* Paper presented at the IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000).

Glover, F. (1977). Heuristics for Integer Programming using Surrogate Constraints. *Decision Sciences, 8*(1), 156-166.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res., 13*(5), 533-549.

Glover, F., e Kochenberger, G. A. (2003). *Handbook of Metaheuristics*: Kluwer Academic Publishers.

Glover, F., e Laguna, M. (1998). *Tabu search*: Kluwer Academic Publishers.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*: Addison-Wesley Longman Publishing Co., Inc.

Goldberg, D., Cicirello, V., Dias, M., Simmons, R., Smith, S., Smith, T., e Stentz, A. (2002). *A Distributed Layered Architecture for Mobile Robot Coordination: Application to Space Exploration*. Paper presented at the 3rd International NASA Workshop on Planning and Scheduling for Space, Houston, TX.

Goldman, B. W., e Tauritz, D. R. (2011). *Meta-evolved empirical evidence of the effectiveness of dynamic parameters.* Paper presented at the 13th annual conference companion on Genetic and evolutionary computation.

Goldman, C. V., e Rosenschein, J. S. (1996). Mutually supervised learning in multiagent systems *Adaption and Learning in Multi-Agent Systems* (pp. 85-96): Springer.

Gonzalez, T. F. (2007). *Handbook of Approximation Algorithms and Metaheuristics*: Taylor & Francis.

Gorodetsky, V., Karsaev, O., Samoylov, V., e Skormin, V. (2008). *Multi-Agent Technology for Air Traffic Control and Incident Management in Airport Airspace*. Paper presented at the AAMAS International Workshop Agents in Traffic and Transportation, Lisboa, Portugal.

Goss, S., Aron, S., Deneubourg, J. L., e Pasteels, J. M. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften, 76*(12), 579-581.

Goswami, R., Ghosh, T. K., e Barman, S. (2011). *Local search based approach in grid scheduling using Simulated Annealing.* Paper presented at the 2nd International Conference on Computer and Communication Technology (ICCCT), 2011

Greenwood, D., Vitaglione, G., Keller, L., e Calisti, M. (2006). *Service Level Agreement Management with Adaptive Coordination.* Paper presented at the International conference on Networking and Services ICNS '06.

Grolimund, S., e Ganascia, J-G. (1997). Driving tabu search with case-based reasoning. *European Journal of Operational Research, 103*(2), 326-338.

Hamadi, Y., Monfroy, E., e Saubion, F. (2012). An Introduction to Autonomous Search. In Y. Hamadi, E. Monfroy e F. Saubion (Eds.), *Autonomous Search* (pp. 1-11): Springer Berlin Heidelberg.

Hansen, N. (2006). The CMA evolution strategy: a comparing review *Towards a new evolutionary computation* (pp. 75-102): Springer.

Harik, G. R., e Lobo, F. G. (1999). *A parameter-less genetic algorithm.* Paper presented at the genetic and evolutionary computation conference.

Hartigan, J. A. (1975). *Clustering algorithms*: John Wiley & Sons, Inc.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics (NRL), 49*(5), 433-448.

Hassan, S., Al-Jumeily, D., e Hussain, A. J. (2009). *Autonomic Computing Paradigm to Support System's Development*. Paper presented at the Second International Conference on Developments in eSystems Engineering.

Hayzelden, A., e Bourne, R. (2001). *Agent technology for communication infrastructures*: John Wiley.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American statistical association, 58*(301), 13-30.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, Michigan, United States: University of Michigan Press.

Hoos, H. H. (2002). *An adaptive noise mechanism for WalkSAT*. Paper presented at the National conference on artificial intelligence.

Horling, B., e Lesser, V. (2004). A survey of multi-agent organizational paradigms. *Knowl. Eng. Rev., 19*(4), 281-316.

Horn, P. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology. *IBM Corporation*.

Hudson, D. L., e Cohen, M. E. (2002). *Use of intelligent agents in the diagnosis of cardiac disorders.* Paper presented at the Computers in Cardiology, 2002.

Huebscher, M. C., e McCann, J. A. (2008). A survey of autonomic computing - degrees, models, and applications. *ACM Comput. Surv., 40*(3), 1-28.

Hutter, F., Hamadi, Y., Hoos, H., e Leyton-Brown, K. (2006a). *Performance prediction and automated tuning of randomized and parametric algorithms*. Paper presented at the 12th international conference on Principles and Practice of Constraint Programming, Nantes, France.

Hutter, F., Hoos, H., e Stutzle, T. (2007). *Automatic algorithm configuration based on local search.* Paper presented at the National Conference on Artificial Intelligence.

Hutter, F., Tompkins, D., e Hoos, H. . (2006b). *Scaling and probabilistic smoothing: Efficient dynamic local search for SAT*. Paper presented at the Principles and Practice of Constraint Programming-CP 2002.

IBM. (2005). An architectural blueprint for autonomic computing *White Paper*.

IBM. (2006). Practical Autonomic Computing: Roadmap to Self Managing Technology *White Paper*.

Jansen, T., e Wiegand, R. P. (2003). *Exploring the explorative advantage of the cooperative coevolutionary (1+ 1) EA.* Paper presented at the Genetic and Evolutionary Computation—GECCO 2003.

Jascanu, N. (2008). *Emotionally based multi-agent e-commerce platform*. Paper presented at the 7th international joint conference on Autonomous agents and multiagent systems: doctoral mentoring program, Estoril, Portugal.

Jen, E. (2003). Stable or robust? What's the difference? *Complexity, 8*(3), 12-18.
Jennings, N. R. (1996). Coordination techniques for distributed artificial intelligence *Foundations of distributed artificial intelligence* (pp. 187-210): John Wiley & Sons, Inc.
Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Commun. ACM, 44*(4), 35-41.
Jennings, N. R., Faratin, P., Lomuscio, A. R., Parsons, S., Wooldridge, M., e Sierra, C. (2001). Automated Negotiation: Prospects, Methods and Challenges. *Group Decision and Negotiation, 10*(2), 199-215.
Jennings, N. R., Sycara, K., e Wooldridge, M. (1998). A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems, 1*(1), 7-38.
Jennings, N. R., e Wooldridge, M. (1998). *Agent Technology: Foundations, Applications, and Markets*: Springer.
Jiang, T., e Tianfield, H. (2009). *Health delivery systems — A case for multi-agent systems.* Paper presented at the IEEE International Conference on Systems, Man and Cybernetics SMC 2009.
Jing-yan, W., e Zhen, Z. (2008). *Framework of multi-agent information retrieval system based on ontology and its application.* Paper presented at the International Conference on Machine Learning and Cybernetics, 2008
Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges, 59*, 215-250.
Kaminsky, D., Miller, B., Salahshour, A., e Whitmore, J. (2008). *Policy-Based Automation in the Autonomic Data Center.* Paper presented at the International Conference on Autonomic Computing ICAC '08.
Kandasamy, N., Abdelwahed, S., e Hayes, J. P. (2004). *Self-optimization in computer systems via on-line control: Application to power management.* Paper presented at the International Conference on Autonomic Computing, 2004.
Karaboga, D. (2005). An Idea Based On Honey Bee Swarm For Numerical Numerical Optimization *Technical Report-TR06*: Erciyes University, Engineering Faculty, Computer Engineering Department.
Karaboga, D., e Akay, B. (2009). A survey: algorithms simulating bee swarm intelligence. *Artificial Intelligence Review, 31*(1-4), 61-85.
Karaboga, D., e Akay, B. (2011). A modified artificial bee colony (ABC) algorithm for constrained optimization problems. *Applied Soft Computing, 11*(3), 3021-3031.
Karaboga, D., e Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization, 39*(3), 459-471.
Kennedy, J., e Eberhart, R. (1995). *Particle swarm optimization.* Paper presented at the IEEE International Conference on Neural Networks.
Kephart, J. O., e Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer, 36*(1), 41-50.
Khargharia, B., Haoting, L., Al-Nashif, Y., e Hariri, S. (2010). *AppFlow: Autonomic Performance-Per-Watt Management of Large-Scale Data Centers.* Paper presented at the Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom).

Khargharia, B., Hariri, S., e Yousif, M. S. (2006). *Autonomic Power and Performance Management for Computing Systems.* Paper presented at the IEEE International Conference on Autonomic Computing ICAC'06.

Kirkpatrick, S., Gelatt, C. D., e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 671-680.

Klusch, M. (2001). Information agent technology for the Internet: a survey. *Data Knowl. Eng., 36*(3), 337-372.

Kolodner, J. (1993). Case-Based Reasoning. *Morgan Kaufmann*, 545-555.

Kose, H., Kaplan, K., Mericli, C., Tatlidede, U., e Akin, L. (2005). *Market-Driven Multi-Agent Collaboration in Robot Soccer Domain*. Paper presented at the Cutting Edge Robotics, Germany.

Kuyer, L., Whiteson, S., Bakker, B., e Vlassis, N. (2008). *Multiagent Reinforcement Learning for Urban Traffic Control Using Coordination Graphs*. Paper presented at the European Conference on Machine Learning and Knowledge Discovery in Databases - Part I, Antwerp, Belgium.

Lagoudakis, M., e Littman, M. (2001). Learning to select branching rules in the DPLL procedure for satisfiability. *Electronic Notes in Discrete Mathematics, 9*, 16-16.

Lalanda, P., McCann, J. A., e Diaconescu, A. (2013). *Autonomic Computing*: Springer.

Lanzola, G., e Boley, H. (2002). Experience with a functional-logic multi-agent architecture for medical problem solving. In G. Rolf e tter (Eds.), *Knowledge media in healthcare* (pp. 17-37): IGI Publishing.

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

Leake, D. B. (1996). *Case-based reasoning: Experiences, lessons and future directions*: MIT press.

Lessmann, S., Caserta, M., e Arango, I. (2011). Tuning metaheuristics: A data mining based approach for particle swarm optimization. *Expert Systems with Applications, 38*(10), 12826-12838.

Leyton-Brown, K., Nudelman, E., e Shoham, Y. (2006). *Learning the empirical hardness of optimization problems: The case of combinatorial auctions.* Paper presented at the Principles and Practice of Constraint Programming-CP 2002.

Li, J-Q., Pan, Q-K., e Liang, Y-C. (2010). An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering, 59*(4), 647-662.

Li, J-Q., Pan, Q-K., Suganthan, P. N., e Chua, T. J. (2011). A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology, 52*(5-8), 683-697.

Liang, D., e Tao, Z. (2011). *Hybrid genetic-Tabu Search approach to scheduling optimization for dual-resource constrained job shop.* Paper presented at the Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC), 2011.

Liao, C-J., Tjandradjaja, E., e Chung, T-P. (2012). An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. *Applied Soft Computing, 12*(6), 1755-1764.

Lin, G. Y., e Solberg, J. J. (1994). *An agent-based flexible routing manufacturing control simulation system.* Paper presented at the Simulation Conference.

Lin, S-W., e Chen, S-C. (2011). Parameter tuning, feature selection and weight assignment of features for case-based reasoning by artificial immune system. *Applied Soft Computing, 11*(8), 5042-5052.

Lin, S-W., Gupta, J., Ying, K-C., e Lee, Z-J. (2009). Using simulated annealing to schedule a flowshop manufacturing cell with sequence-dependent family setup times. *International Journal of Production Research, 47*(12), 3205-3217.

Liu, H., Abraham, A., e Hassanien, A. (2010). Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. *Future Generation Computer Systems, 26*(8), 1336-1343.

Lobo, F. G., Lima, C. F., e Michalewicz, Z. (2007). *Parameter setting in evolutionary algorithms* (Vol. 54): Springer Verlag.

Luck, M., McBurney, P., Shehory, O., e Willmott, S. (2005). *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*: University of Southampton.

Luke, S. (2012). *Essentials of Metaheuristics* (First Edition (Rev. C), online version 1.3 ed.): Lulu.com.

Machado, C. (2009). *Autonomic Ubiquitous Computing: A Home Environment Management System.* (PhD thesis), Universidade do Minho, Portugal.

Madureira, A. (2003). *Aplicação de Meta-Heurísticas ao Problema de Escalonamento em Ambiente Dinâmico de Produção Discreta.* (PhD thesis), Universidade do Minho, Braga, Portugal.

Madureira, A. (2010). *Técnicas Emergentes de Optimização no Suporte à Tomada de Decisão*. Instituto Superior de Engenharia do Porto.

Madureira, A., Cunha, B., e Pereira, I. (2014). *Cooperation Mechanism for Distributed Resource Scheduling for Artificial Bee Colony based Self-Organized Scheduling System.* Paper presented at the 2014 IEEE Congress on Evolutionary Computation, Beijing, China.

Madureira, A., Pereira, I., e Falcão, D. (2013a). Cooperative Scheduling System with Emergent Swarm Based Behavior *Advances in Information Systems and Technologies* (pp. 661-671): Springer.

Madureira, A., Pereira, I., e Falcão, D. (2013b, September 09-10). *Dynamic Adaptation for Scheduling Under Rush Manufacturing Orders With Case-Based Reasoning.* Paper presented at the International Conference on Algebraic and Symbolic Computation (SymComp 2013), Lisbon, Portugal.

Madureira, A., Pereira, I., Pereira, P., e Abraham, A. (2013c). Negotiation mechanism for self-organized scheduling system with collective intelligence. *Neurocomputing, Volume 132*, 97-110.

Madureira, A., Pereira, I., e Sousa, N. (2010a). *Collective intelligence on dynamic manufacturing scheduling optimization.* Paper presented at the IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA).

Madureira, A., Pereira, I., e Sousa, N. (2011a). *Self-organization for scheduling in agile manufacturing.* Paper presented at the IEEE 10th International Conference on Cybernetic Intelligent Systems (CIS).

Madureira, A., Pereira, I., Sousa, N., Ávila, P., e Bastos, J. (2009a). *Autonomic Computing for Scheduling in Manufacturing Systems*. Paper presented at the International Symposium on Computational Intelligence for Engineering Systems (ISCIES09), Instituto Superior de Engenharia do Porto (ISEP), Porto, Portugal.

Madureira, A., Santos, F., e Pereira, I. (2008). *Self-managing agents for dynamic scheduling in manufacturing.* Paper presented at the 10th annual conference companion on Genetic and evolutionary computation (GECCO'08), New York, USA.

Madureira, A., Santos, J., e Pereira, I. (2009b). MASDScheGATS – Scheduling System for Dynamic Manufacturing Environments. *MultiAgent Systems, In-Tech*.

Madureira, A., Santos, J., e Pereira, I. (2007). *MASDScheGATS: a prototype system for dynamic scheduling.* Paper presented at the 6th WSEAS international conference on Computational intelligence, man-machine systems and cybernetics.

Madureira, A., Santos, J., e Pereira, I. (2009c). A Hybrid Intelligent System for Distributed Dynamic Scheduling. In R. Chiong e S. Dhakal (Eds.), *Natural Intelligence for Scheduling, Planning and Packing Problems* (Vol. 250, pp. 295-324): Springer Berlin Heidelberg.

Madureira, A., Sousa, N., e Pereira, I. (2010b). *Mecanismo de Negociação para Sistema de Escalonamento Dinâmico*. Paper presented at the WACI'10 - 5th Workshop on Applications of Computational Intelligence, Coimbra, Portugal.

Madureira, A., Sousa, N., e Pereira, I. (2011b). *Negotiation mechanism for self-organized scheduling system.* Paper presented at the Third World Congress on Nature and Biologically Inspired Computing (NaBIC).

Maes, P. (1995). Artificial life meets entertainment: lifelike autonomous agents. *Commun. ACM, 38*(11), 108-114.

Malone, T. W., e Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Comput. Surv., 26*(1), 87-119.

Maniezzo, V., Gambardella, L. M. , e De Luigi, F. (2004). Ant Colony Optimization *New Optimization Techniques in Engineering* (pp. 101-117): Springer.

Maron, O. (1994). *Hoeffding Races: Model Selection for MRI Classification.* Massachusetts Institute of Technology, Cambridge, Massachusetts, United States of America.

Maron, O., e Moore, A. W. (1993). Hoeffding races: Accelerating model selection search for classification and function approximation. *Robotics Institute*, 263.

Maron, O., e Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review, 11*(1-5), 193-225.

Matossian, V., Bhat, V., Parashar, M., Peszynska, M., Sen, M., Stoffa, P., e Wheeler, M. F. (2005). Autonomic oil reservoir optimization on the Grid: Research Articles. *Concurr. Comput. : Pract. Exper., 17*(1), 1-26.

Maturana, F. P. (1997). *MetaMorph: An Adaptive Multi-Agent Architecture for Advanced Manufacturing Systems.* (PhD thesis), University of Calgary, Calgary, Alberta, Canada.

Maturana, F. P., Shen, W., e Norrie, D. H. (1999). MetaMorph: an adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research, 37*(10), 2159-2173.

McCann, J. A., e Sterritt, R. (2010). *Autonomic Pervasive Networks (APNs) - Extended Abstract*. Paper presented at the Seventh IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems.

McKay, M. D., Beckman, R. J., e Conover, W. J. (1979). Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics, 21*(2), 239-245.

Mendenhall, W., Scheaffer, R. L., e Wackerly, D. D. (1981). *Mathematical statistics with applications*: Duxbury Press Boston.

Mendes, J. J. M., Gonçalves, J. F., e Resende, M. G. C. (2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research, 36*(1), 92-109.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., e Teller, E. (1953). Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics, 21*, 1087-1092.

Mika, M., Rozycki, R., e Waligora, G. (2011). *Local search metaheuristics for some discrete-continuous project scheduling problems with discounted cash flows.* Paper presented at the 16th International Conference on Methods and Models in Automation and Robotics (MMAR).

Minsky, M. (1986). *The society of mind*: Simon & Schuster, Inc.

Mitchell, T. (1997). *Machine Learning*: McGraw-Hill Education (ISE Editions).

Montgomery, D. C. (2008). *Design and analysis of experiments*: Wiley.

Moore, A. W., e Lee, M. S. (1994). *Efficient Algorithms for Minimizing Cross Validation Error.* Paper presented at the ICML.

Moreno, A., e Nealon, J. L. (2004). *Applications of Software Agent Technology in the Health Care Domain*: Birkhauser (Architectural).

Moscato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts – Towards Memetic Algorithms. In C. C. C. Program (Ed.), *C3P Report 826*.

Moslehi, G., e Mahnam, Mehdi. (2011). A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. *International Journal of Production Economics, 129*(1), 14-22.

Naderi, B., Tavakkoli-Moghaddam, R., e Khalili, M. (2010). Electromagnetism-like mechanism and simulated annealing algorithms for flowshop scheduling problems minimizing the total weighted tardiness and makespan. *Knowledge-Based Systems, 23*(2), 77-85.

Nami, M. R., e Sharifi, M. (2007). *Autonomic Computing: A New Approach.* Paper presented at the First Asia International Conference on Modelling & Simulation AMS '07.

Nannen, V., e Eiben, A. E. (2006). *A method for parameter calibration and relevance estimation in evolutionary algorithms.* Paper presented at the 8th annual conference on Genetic and evolutionary computation.

Nannen, V., e Eiben, A. E. (2007). *Relevance estimation and value calibration of evolutionary algorithm parameters.* Paper presented at the 20th international joint conference on Artifical intelligence.

Nannen, V., Smit, S. K., e Eiben, A. E. (2008). Costs and benefits of tuning parameters of evolutionary algorithms *Parallel Problem Solving from Nature–PPSN X* (pp. 528-538): Springer.

Nilsson, N. (1981). Distributed Artificial Intelligence *SRI International*. Menlo Park, California, United States.

Nouyan, S. (2008). *Teamwork in a Swarm of Robots – An Experiment in Search and Retrieval.* (PhD thesis), Universite Libre de Bruxelles, Brussels, Belgium.

Nudelman, E., Leyton-Brown, K., Hoos, H. H, Devkar, A., e Shoham, Y. (2004). Understanding random SAT: Beyond the clauses-to-variables ratio *Principles and Practice of Constraint Programming–CP 2004* (pp. 438-452): Springer.

Nwana, H. S., Lee, L., e Jennings, N.R. (1996). Coordination in Software Agent Systems. *BT Technology Journal, 14*(4), 79-88.

Oltean, M. (2005). Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation, 13*(3), 387-410.

Oman, S., e Cunningham, P. (2001). Using case retrieval to seed genetic algorithms. *International Journal of Computational Intelligence and Applications, 1*(01), 71-82.

Osman, I., e Kelly, J. (1996). *Meta-Heuristics: Theory and Applications*: Kluwer Academic Publishers.

Ouelhadj, D., Hanachi, C., e Bouzouia, B. (2000, 2000). *Multi-agent architecture for distributed monitoring in flexible manufacturing systems (FMS).* Paper presented at the IEEE International Conference on Robotics and Automation, ICRA '00.

Ouelhadj, D., e Petrovic, S. (2008). A Survey of Dynamic Scheduling in Manufacturing Systems. *Journal of Scheduling*.

Pan, Q-K., Suganthan, P. N., Tasgetiren, M. F., e Liang, J. J. (2010). A self-adaptive global best harmony search algorithm for continuous optimization problems. *Applied Mathematics and Computation, 216*(3), 830-848.

Pan, Q-K., Tasgetiren, M. F., Suganthan, P. N., e Chua, T. J. (2011). A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information Sciences, 181*(12), 2455-2468.

Panait, L., e Luke, S. (2005). Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems, 11*(3), 387-434.

Pansuwan, P., Rukwong, N., e Pongcharoen, P. (2010). *Identifying Optimum Artificial Bee Colony (ABC) Algorithm's Parameters for Scheduling the Manufacture and Assembly of Complex Products.* Paper presented at the Second International Conference on Computer and Network Technology (ICCNT).

Paolino, L., Paggi, H., Alonso, F., e Lopez, G. (2011). *Solving incidents in telecommunications using a multi-agent system.* Paper presented at the IEEE International Conference on Intelligence and Security Informatics (ISI).

Parashar, M. (2006). Autonomic Grid Computing *Autonomic Computing* (pp. 49-70): CRC Press.

Parashar, M., e Hariri, S. (2006). *Autonomic Computing: Concepts, Infrastructure, and Applications*: Taylor & Francis.

Pavón, R., Díaz, F., Laza, R., e Luzón, V. (2009). Automatic parameter tuning with a Bayesian case-based reasoning system. A case of study. *Expert Systems With Applications, 36*(2), 3407-3420.

Peizhao, H., Indulska, J., e Robinson, R. (2008). *An Autonomic Context Management System for Pervasive Computing.* Paper presented at the Sixth Annual IEEE International Conference on Pervasive Computing and Communications PerCom 2008.

Pereira, I. (2009). *Aspectos de Aprendizagem em Optimização.* (MSc thesis), Instituto Superior de Engenharia do Porto, Porto, Portugal.

Pereira, I., e Madureira, A. (2013). Self-Optimization module for Scheduling using Case-based Reasoning. *Applied Soft Computing, 13*(3), 1419-1432.

Pereira, I., Madureira, A., e de Moura Oliveira, P. (2012, 5-9 Nov. 2012). *Multi-apprentice learning for meta-heuristics parameter tuning in a Multi Agent Scheduling System.* Paper presented at the Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC).

Pereira, I., Madureira, A., e Moura Oliveira, P. (2013a). Meta-heuristics Self-Parameterization in a Multi-agent Scheduling System Using Case-Based Reasoning. In A. Madureira, C. Reis e V. Marques (Eds.), *Computational Intelligence and Decision Making* (Vol. 61, pp. 99-109): Springer Netherlands.

Pereira, I., Madureira, A., Oliveira, P., e Abraham, A. (2013b). Tuning Meta-heuristics Using Multi-agent Learning in a Scheduling System. *Transactions on Computational Science, Springer Verlag*.

Peterson, L. (2011). Covariance matrix self-adaptation evolution strategies and other metaheuristic techniques for neural adaptive learning. *Soft Computing, 15*(8), 1483-1495.

Petric, A. (2008). *A Multi-Agent System for Content Trading in Electronic Telecom Markets Using Multi-Attribute Auctions*. Paper presented at the 10th International Conference on Electronic Commerce (ICEC 2008).

Petrovic, S., Yang, Y., e Dror, M. (2007). Case-based selection of initialisation heuristics for metaheuristic examination timetabling. *Expert Systems with Applications, 33*(3), 772-785.

Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., e Zaidi, M. (2005). The Bees Algorithm. In M. E. Centre (Ed.), *Technical Note*. Cardiff, Wales, United Kingdom: Cardiff University.

Pham, D. T., e Karaboga, D. (1998). *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*: Springer-Verlag New York, Inc.

Pinedo, M. (2008). *Scheduling: Theory, Algorithms, and Systems*: Springer.

Pinto, T. (2011). *Adaptive Learning in Agents Behaviour: a Framework for Electricity Markets Simulation.* (Tese de Mestrado em Engenharia Informática, Ramo de Tecnologias do Conhecimento e Decisão), Instituto Superior de Engenharia do Porto.

Pirlot, M. (1996). General local search methods. *European Journal of Operational Research, 92*(3), 493-511.

Plaza, E., Arcos, J. L., e Martin, F. (1997). Cooperative case-based reasoning *Distributed Artificial Intelligence Meets Machine Learning Learning in Multi-Agent Environments* (pp. 180-201): Springer.

Pongchairerks, P., e Kachitvichyanukul, V. (2009). A particle swarm optimization algorithm on job-shop scheduling problems with multi-purpose machines. *Asia-Pacific Journal of Operational Research, 26*(02), 161-184.

Porter, B. W., e Bareiss, E. R. (1986). PROTOS: An Experiment in Knowledge Acquisition for Heuristic ClassificationTasks. Austin, Texas, United States: University of Texas.

Portmann, M. C. (1997). Scheduling Methodology: optimization and compu-search approaches *The planning and scheduling of production systems*: Chapman & Hall.

Prot, D., e Bellenguez-Morineau, O. (2012). Tabu search and lower bound for an industrial complex shop scheduling problem. *Computers & Industrial Engineering, 62*(4), 1109-1118.

Qiang, W. (2009). *An Efficient Data Mining System Using Multi-agent of e-Commerce.* Paper presented at the 7th ACIS International Conference on Software Engineering Research, Management and Applications SERA '09.

Qiao, L., Ning, S., e Shibing, Z. (2010). *A New Research of Resource Scheduling Based on Multi-agent Manufacturing Grid.* Paper presented at the Asia-Pacific Conference on Wearable Computing Systems (APWCS).

Qin, A. K., Huang, V. L., e Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation, 13*(2), 398-417.

Qing-dao-er-ji, R., e Wang, Y. (2012). A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research, 39*(10), 2291-2299.

Rabiner, L., e Juang, B-H. (1986). An introduction to hidden Markov models. *ASSP Magazine, IEEE, 3*(1), 4-16.

Ramirez, A. E., Morales, B., e King, T. M. (2008). *A self-testing autonomic job scheduler*. Paper presented at the 46th Annual Southeast Regional Conference on XX, Auburn, Alabama.

Ranganathan, A. (2004). *Autonomic Pervasive Computing Based on Planning*. Paper presented at the First International Conference on Autonomic Computing.

Ranganathan, A., Shankar, C., e Campbell, R. (2005). Application polymorphism for autonomic ubiquitous computing. *Multiagent Grid Syst., 1*(2), 109-129.

Reeves, C. (1993). *Modern heuristic techniques for combinatorial problems*: John Wiley & Sons, Inc.

Reis, L. P. (2002). Coordenação de Agentes Cooperativos e Trabalho de Equipa *Research Report*: LIACC (NIAD&R).

Reis, L. P. (2003). *Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico.* (PhD thesis), Faculdade de Engenharia da Universidade do Porto, Porto, Portugal.

Rice, J. R. (1975). The algorithm selection problem *Technical Report nº 75-152*. West Lafayette, Indiana, United States: Purdue University.

Rodero, I., Jaramillo, J., Quiroz, A., Parashar, M., Guim, F., e Poole, S. (2010). *Energy-efficient application-aware online provisioning for virtualized clouds and data centers.* Paper presented at the International Green Computing Conference.

Ross, B. H. (1989). *Some psychological results on case-based reasoning.*

Ross, K., e Bambos, N. (2006). *Job scheduling for maximal throughput in autonomic computing systems*. Paper presented at the First international conference and Third international conference on New Trends in Network Architectures and Services conference on Self-Organising Systems, Passau, Germany.

Rostami, V., Ebrahimijam, S., khajehpoor, P., Mirzaei, P., e Yousefiazar, M. (2005). *Cooperative multi agent soccer robot team*. Paper presented at the World Academy of Science, Engineering and Technology.

Rudolph, G., Preuss, M., e Quadflieg, J. (2009). Two-layered surrogate modeling for tuning optimization metaheuristics *Algorithm Engineering report*. Dortmund, Germany: Technische Universität.

Ruimei, L. (2011). *Construction of medicine e-commerce system based on Multi-Agent.* Paper presented at the International Conference on Computer Science and Service System (CSSS).

Russell, S. J., e Norvig, P. (2010). *Artificial intelligence: a modern approach*: Pearson Education/Prentice Hall.

Sammut, C., e Webb, G. I. (2011). *Encyclopedia of machine learning*: Springer-Verlag New York Incorporated.

Sandholm, T. (2000). *eMediator: a next generation electronic commerce server*. Paper presented at the Fourth international conference on Autonomous agents, Barcelona, Spain.

Sandholm, T., e Crites, R. (1996). On multiagent Q-learning in a semi-competitive domain *Adaption and Learning in Multi-Agent Systems* (pp. 191-205): Springer.

Santner, T. J., Williams, B. J., e Notz, W. I. (2003). *The design and analysis of computer experiments*: Springer Verlag.

Schaffer, J. D., Caruana, R. A., Eshelman, L. J., e Das, R. (1989). *A study of control parameters affecting online performance of genetic algorithms for function optimization.* Paper presented at the Third international conference on Genetic algorithms.

Schank, R. (1982). *Dynamic memory: a theory of reminding and learning in computers and people*: Cambrige University Press.

Scheutz, M., e Schermerhorn, P. (2003). *Many is more, but not too many: dimensions of cooperation of agents with and without predictive capabilities.* Paper presented at the IEEE/WIC International Conference on Intelligent Agent Technology IAT 2003.

Schirmer, A. (2000). Case-based reasoning and improved adaptive search for project scheduling. *Naval Research Logistics (NRL), 47*(3), 201-222.

Schmidhuber, J. (1996). *Realistic multi-agent reinforcement learning.* Paper presented at the Learning in Distributed Artificial Intelligence Systems. Working Notes of the 1996 ECAI Workshop.

Schmidhuber, J., e Zhao, J. (1997). Multi-agent learning with the success-story algorithm *Distributed Artificial Intelligence Meets Machine Learning Learning in Multi-Agent Environments* (pp. 82-93): Springer.

Schmidt, G. (1998). Case-based reasoning for production scheduling. *International Journal of Production Economics, 56*, 537-546.

Shanhong, Z., Wanlong, L., Xinyi, P., Hui, Z., e Chunfei, Z. (2010). *Research on integrating the healthcare enterprise based on multi-agent.* Paper presented at the International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE).

Shen, W., Maturana, F. P., e Norrie, D. H. (2000). Metaphor II: an agent-based architecture for distributed intelligent design and manufacturing. *Journal of Intelligent Manufacturing, 11*(3), 237-251.

Shen, W., e Norrie, D. H. (1999). Agent based systems for intelligent manufacturing: a state of the art survey. *International Journal of Knowledge and Information Systems, 1*(2), 129-156.

Shen, W., Norrie, D. H., e Barthes, J. P. (2004). *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*: Taylor & Francis.

Sheskin, D. J. (2003). *Handbook of parametric and nonparametric statistical procedures*: CRC Press.

Shooter, S. B., Simpson, T. W., Kumara, S. R. T., Stone, R., e Terpenny, J. (2005). Toward a Multi-Agent Information Management Infrastructure for Product Family Planning and Mass Customisation. *International Journal of Mass Customisation*.

Singh, B. (1992). A Coordination Model *Technical Report CT084-92*. Austin, Texas, United States of America: Microelectronics and Computer Technology Corp. (MCC).

Smit, S. K., e Eiben, A. E. (2009). *Comparing parameter tuning methods for evolutionary algorithms.* Paper presented at the IEEE Congress on Evolutionary Computation CEC'09.

Smit, S. K., e Eiben, A. E. (2010). Using entropy for parameter analysis of evolutionary algorithms *Experimental Methods for the Analysis of Optimization Algorithms* (pp. 287-310): Springer.

Smith-Miles, K. (2008). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR), 41*(1), 6.

Smith, D. C., Cypher, A., e Spohrer, J. (1994). KidSim: programming agents without a programming language. *Commun. ACM, 37*(7), 54-67.

Smith, J. E. (2008). Self-adaptation in evolutionary algorithms for combinatorial optimisation *Adaptive and Multilevel Metaheuristics* (pp. 31-57): Springer.

Smith, R. G. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers, C-29*(12), 1104-1113.

Socha, K. (2008). *Ant Colony Optimization for Continuous and Mixed-Variable Domains.* (PhD thesis), Universit´e Libre de Bruxelles, Brussels, Belgium.

Socha, K. (2009). *Ant Colony Optimisation for Continuous and Mixed-variable Domains*: VDM Publishing.

Song, S., Ren, J., e Fan, J. (2012). Improved Simulated Annealing Algorithm Used for Job Shop Scheduling Problems. In A. Xie e X. Huang (Eds.), *Advances in Electrical Engineering and Automation* (Vol. 139, pp. 17-25): Springer Berlin Heidelberg.

Stoean, R., Bartz-Beielstein, T., Preuss, M., e Stoean, C. (2009). A Support Vector Machine-Inspired Evolutionary Approach for Parameter Tuning in Metaheuristics.

Stone, P. (2000). *Layered learning in multiagent systems: a winning approach to robotic soccer*: "The" MIT Press.

Stone, P., e Veloso, M. (1998). Towards collaborative and adversarial learning:: a case study in robotic soccer. *Int. J. Hum.-Comput. Stud., 48*(1), 83-104.

Stone, P., e Veloso, M. (2000). Multiagent Systems: A Survey from a Machine Learning Perspective. *Auton. Robots, 8*(3), 345-383.

Storer, R., Wu, S. D., e Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management science, 38*(10), 1495-1509.

Storn, R., e Price, K. (1997). Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. of Global Optimization, 11*(4), 341-359.

Stützle, T., López-Ibánez, M., Pellegrini, P., Maur, M., de Oca, M. M., Birattari, M., e Dorigo, M. (2012). Parameter adaptation in ant colony optimization *Autonomous Search* (pp. 191-215): Springer.

Sutton, R. S., e Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1): Cambridge Univ Press.

T. Yamada, e Nakano, R. (1992). A genetic algorithm applicable to large-scale job-shop instances. *Parallel instance solving from nature 2*, 281-290.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*(2), 278-285.

Talbi, E.G. (2009). *Metaheuristics: From Design to Implementation*: Wiley.

Tasgetiren, M. F., Bulut, O., e Fadiloglu, M. M. (2011). *A discrete artificial bee colony algorithm for the economic lot scheduling problem.* Paper presented at the IEEE Congress on Evolutionary Computation (CEC).

Tavares Neto, R. F., e Godinho Filho, M. (2011). An ant colony optimization approach to a permutational flowshop scheduling problem with outsourcing allowed. *Computers & Operations Research, 38*(9), 1286-1293.

Telser, L. G. (1987). *A Theory of Effective Cooperation and Competition*. Cambridge: Cambridge University Press.

Toriz, A., Sánchez, A., e Osorio, M. (2009). Coordinated multi-agent exploration. *Journal Research in Computing Science, 42*, 189-200.

Tseng, L-Y., e Lin, Y-T. (2009). A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 198*(1), 84-92.

Vallada, E., e Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research, 211*(3), 612-622.

Vapnik, V. (1998). *Statistical learning theory* (Vol. 2): Wiley New York.

Vela, C. R., Varela, R., e González, M. A. (2010). Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics, 16*(2), 139-165.

Viamonte, M. J., Ramos, C., Rodrigues, F., e Cardoso, J. C. (2006). ISEM: a multiagent Simulator for testing agent market strategies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 36*(1), 107-113.

Viana, A. (2003). Agentes Inteligentes e Sistemas Multi-agente FIPA. Lisboa: Instituto Superior Técnico.

Vilcot, G., e Billaut, J-C. (2008). A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem. *European Journal of Operational Research, 190*(2), 398-411.

Wang, H-M., Chou, F-D., e Wu, F-C. (2011a). A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan. *The International Journal of Advanced Manufacturing Technology, 53*(5-8), 761-776.

Wang, P., Chen, Y., Chen, S., e Ye, G. (2011b). *ISRMDSS: An information security risk management oriented multi-agent system.* Paper presented at the IEEE 3rd International Conference on Communication Software and Networks (ICCSN).

Wang, X., e Tang, L. (2009). A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers. *Comput. Oper. Res., 36*(3), 907-918.

Watkins, C., e Dayan, P. (1992). Q-learning. *Machine learning, 8*(3-4), 279-292.

Watson, J-P., e Beck, J. C. (2008). *A hybrid constraint programming/local search approach to the job-shop scheduling problem*. Paper presented at the 5th international conference on Integration of AI and OR techniques in constraint programming for combinatorial optimization problems, Paris, France.

Weinberg, M., e Rosenschein, J. S. (2004). *Best-response multiagent learning in non-stationary environments.* Paper presented at the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2.

Weiß, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*: Mit Press.

Whiteson, S., e Stone, P. (2004). *Towards autonomic computing: adaptive job routing and scheduling*. Paper presented at the 16th conference on Innovative applications of artifical intelligence, San Jose, California.

Williams, A. B. (2004). Learning to share meaning in a multi-agent system. *Autonomous Agents and Multi-Agent Systems, 8*(2), 165-193.

Wolpert, D. H., e Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67-82.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*: John Wiley & Sons Australia, Limited.

Wooldridge, M., e Jennings, N. (1995). Agent theories, architectures, and languages: A survey. In M. Wooldridge e N. Jennings (Eds.), *Intelligent Agents* (Vol. 890, pp. 1-39): Springer Berlin Heidelberg.

Xiang, W., e Lee, H. P. (2008). Ant colony intelligence in multi-agent dynamic manufacturing scheduling. *Engineering Applications of Artificial Intelligence, 21*(1), 73-85.

Xiangliang, Z., Germain, C., e Sebag, M. (2010). *Adaptively detecting changes in Autonomic Grid Computing.* Paper presented at the 11th IEEE/ACM International Conference on Grid Computing (GRID).

Xiao, R., e Chen, T. (2011). *Enhancing abc optimization with ai-net algorithm for solving project scheduling problem.* Paper presented at the Seventh International Conference on Natural Computation (ICNC).

Xing, L-N., Chen, Y-W., Wang, P., Zhao, Q-S., e Xiong, J. (2010). A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing, 10*(3), 888-896.

Xu, L., Hutter, F., Hoos, H., e Leyton-Brown, K. (2008). SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research, 32*(1), 565-606.

Xu, R., Chen, H., e Li, X. (2012). Makespan minimization on single batch-processing machine via ant colony optimization. *Computers & Operations Research, 39*(3), 582-593.

Yagmahan, B., e Yenisey, M. M. (2010). A multi-objective ant colony system algorithm for flow shop scheduling problem. *Expert Systems with Applications, 37*(2), 1361-1368.

Yang, Y. (2004). *Case Based Selection of Initialisation Heuristics for Metaheuristic Examination Timetabling.* (PhD thesis), The University of Nottingham, United Kingdom.

Yang, Z., Tang, K., e Yao, X. (2008). *Self-adaptive differential evolution with neighborhood search.* Paper presented at the IEEE World Congress on Computational Intelligence Evolutionary Computation (CEC 2008).

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

Yoshikawa, M., e Terai, Hi. (2006). *A hybrid ant colony optimization technique for job-shop scheduling problems.* Paper presented at the IEEE/ACIS International Conference on Software Engineering Research, Management & Applications.

Yuan, B., e Gallagher, M. (2004). *Statistical racing techniques for improved empirical evaluation of evolutionary algorithms.* Paper presented at the Parallel Problem Solving from Nature-PPSN VIII.

Yuan, B., e Gallagher, M. (2007). Combining Meta-EAs and racing for difficult EA parameter tuning tasks *Parameter Setting in Evolutionary Algorithms* (pp. 121-142): Springer.

Zelenka, J. (2011). *Parallel computing application into the particle swarm optimization algorithm used to solve the Job-Shop scheduling problem.* Paper presented at the 15th IEEE International Conference on Intelligent Engineering Systems (INES).

Zelenka, J., e Kasanicky, T. (2011). *Comparison of artificial immune systems with the particle swarm optimization in job-shop scheduling problem.* Paper presented at the IEEE 9th International Symposium on Applied Machine Intelligence and Informatics (SAMI).

Zennaki, M., e Ech-Cherif, A. (2010). A New Machine Learning based Approach for Tuning Metaheuristics for the Solution of Hard Combinatorial Optimization Problems. *Journal of Applied Sciences(Faisalabad), 10*(18), 1991-2000.

Zhang, H-T., Yu, F., e Li, W. (2009). Step-coordination algorithm of traffic control based on multi-agent system. *International Journal of Automation and Computing, 6*(3), 308-313.

Zhang, R., Song, S., e Wu, C. (2012). A hybrid artificial bee colony algorithm for the job shop scheduling problem. *International Journal of Production Economics*.

Zhang, R., e Wu, C. (2010). A hybrid immune simulated annealing algorithm for the job shop scheduling problem. *Applied Soft Computing, 10*(1), 79-89.

Zhang, R., e Wu, C. (2012). A hybrid local search algorithm for scheduling real-world job shops with batch-wise pending due dates. *Eng. Appl. Artif. Intell., 25*(2), 209-221.

Zhou, G., Wang, L., Xu, Y., e Wang, S. (2012). An effective artificial bee colony algorithm for multi-objective flexible job-shop scheduling problem *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence* (pp. 1-8): Springer.

Zhou, H., Cheung, W., e Leung, L. C. (2009). Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. *European Journal of Operational Research, 194*(3), 637-649.

Zhu, J., Li, X., e Shen, W. (2011). Effective genetic algorithm for resource-constrained project scheduling with limited preemptions. *International Journal of Machine Learning and Cybernetics, 2*(2), 55-65.