Instituto Superior de Engenharia do Porto

ASPECTOS DE APRENDIZAGEM EM OPTIMIZAÇÃO

Ivo André Soares Pereira

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Área de Especialização em
Tecnologias do Conhecimento e Decisão

Orientador: Doutora Ana Maria Dias Madureira Pereira

Júri:

Presidente:

Doutora Maria de Fátima Coutinho Rodrigues, Professora Coordenadora, Instituto Superior de Engenharia do Porto, Departamento de Engenharia Informática

Vogais:

Doutor Francisco José Baptista Pereira, Professor Adjunto, Instituto Superior de Engenharia de Coimbra, Departamento de Engenharia Informática e de Sistemas

Doutor Paulo António da Silva Ávila, Professor Coordenador, Instituto Superior de Engenharia do Porto, Departamento de Engenharia Mecânica

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

Porto, Outubro de 2009

«À minha família e amigos»

AGRADECIMENTOS

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

Começo por agradecer à minha orientadora, **Doutora Ana Maria Dias Madureira Pereira** pela compreensão e disponibilidade que sempre demonstrou, pelos conselhos e críticas construtivas, e sobretudo pelo enorme trabalho de revisão desta dissertação.

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

Aos meus **colegas de curso**, em especial ao Rúben, ao Bruno, e à Diana, por todo o apoio, sugestões, críticas, companheirismo e amizade que demonstraram ao longo dos últimos dois anos, e aos restantes elementos da *TCDTeam*, pelos momentos de descontracção que me proporcionaram.

A todos os meus **colegas do GECAD**, em especial ao João, ao Paulo, e ao Hélio, pela ajuda e apoio em várias questões em torno deste trabalho, e ao Joaquim, que, na qualidade de administrador de sistemas e redes, deu sempre o seu melhor para que as condições de trabalho fossem as melhores.

Aos **docentes** que, ao longo destes dois últimos anos, contribuíram na evolução dos meus conhecimentos.

À minha **família**, em especial aos meus pais e ao meu irmão, por todo o apoio e compreensão ao longo deste último ano.

A todos os meus **amigos** por todo o apoio demonstrado e pela paciência e compreensão nos momentos mais difíceis.

A todos, o meu muito obrigado!

RESUMO

A optimização e a aprendizagem em Sistemas Multi-Agente são consideradas duas áreas promissoras mas relativamente pouco exploradas. A optimização nestes ambientes deve ser capaz de lidar com o dinamismo. Os agentes podem alterar o seu comportamento baseando-se em aprendizagem recente ou em objectivos de optimização. As estratégias de aprendizagem podem melhorar o desempenho do sistema, dotando os agentes da capacidade de aprender, por exemplo, qual a técnica de optimização é mais adequada para a resolução de uma classe particular de problemas, ou qual a parametrização é mais adequada em determinado cenário.

Nesta dissertação são estudadas algumas técnicas de resolução de problemas de Optimização Combinatória, sobretudo as Meta-heurísticas, e é efectuada uma revisão do estado da arte de Aprendizagem em Sistemas Multi-Agente. É também proposto um módulo de aprendizagem para a resolução de novos problemas de escalonamento, com base em experiência anterior.

O módulo de Auto-Optimização desenvolvido, inspirado na Computação Autónoma, permite ao sistema a selecção automática da Meta-heurística a usar no processo de optimização, assim como a respectiva parametrização. Para tal, recorreu-se à utilização de Raciocínio baseado em Casos de modo que o sistema resultante seja capaz de aprender com a experiência adquirida na resolução de problemas similares.

Dos resultados obtidos é possível concluir da vantagem da sua utilização e respectiva capacidade de adaptação a novos e eventuais cenários.

Palavras-chave: Aprendizagem, Optimização, Escalonamento, Meta-heurísticas, Sistemas Multi-Agente, Raciocínio baseado em Casos, Computação Autónoma

ABSTRACT

Optimization and learning in Multi-Agent Systems are considered two promising areas but relatively unexplored, and optimization in these environments should be capable of dealing with dynamism. Agents can change their behaviours based on recent learning or based on optimization objectives. Learning strategies can improve the system's performance, endowing agents with the ability to learn, for example, which optimization technique is more appropriate for the resolution of a particular class of problems, or which parameterization is more suitable to a given scenario.

In this dissertation are studied some Combinatorial Optimization problems' resolution techniques, especially Meta-heuristics, and it is made a revision of Multi-Agent Systems' Learning state-of-art. It is also proposed a learning module for the resolution of new scheduling problems, based on past experience.

The developed Self-Optimizing module, inspired on Autonomic Computing, allows the system to automatically select the Meta-heuristic to use in the optimization process, so as its parameterization. For that, Case-based Reasoning was used so the resulting system can be capable of learning about the acquired experience, in the resolution of similar problems.

From the obtained results it is possible to conclude about the benefit of its use and its capacity of adapting to new and potential scenarios.

Keywords: Learning, Optimization, Scheduling, Meta-heuristics, Multi-Agent Systems, Case-based Reasoning, Autonomic Computing

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1. INTRODUÇÃO

1.1. Enquadramento e motivação

Os problemas de Optimização Combinatória surgem quando existe a necessidade de se seleccionar, 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 óptimas.

Os problemas de Escalonamento são classificados como problemas de Optimização Combinatória sujeitos a restrições, com uma natureza dinâmica e de resolução muito complexa, sendo classificados como *NP-difíceis*, cujos elementos básicos são as máquinas e as tarefas. O Escalonamento visa a afectação 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.

De entre as várias abordagens de resolução de problemas de Optimização Combinatória, onde se inclui o problema de Escalonamento, salientam-se os métodos de aproximação, nos quais o objectivo passa por encontrar soluções satisfatórias em tempos de execução aceitáveis. Dentro destes métodos incluem-se as Meta-heurísticas, que consistem em métodos inspirados na natureza para pesquisa de soluções o mais próximo possível do óptimo global de um determinado problema. A aplicação destas técnicas necessita de um conhecimento prévio dos parâmetros mais adequados ao problema a tratar, tarefa bastante difícil e que requer normalmente alguma experiência e conhecimento pericial.

Esta dissertação de mestrado foi realizada no âmbito do Projecto de I&D, aprovado pela Fundação para a Ciência e a Tecnologia, AutoDynAgents – *Autonomic Agents with Self-Managing Capabilities for Dynamic Scheduling Support in a Cooperative Manufacturing System* (POCTI/EME-GIN/66848/2006).

O AutoDynAgents consiste num Sistema Multi-Agente 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-óptimos. Uma vez que o ambiente de actuação do AutoDynAgents é complexo, dinâmico e imprevisível, as questões de aprendizagem tornaram-se imprescindíveis.

Assim, surgiu a necessidade de implementação de um módulo de Auto-Optimização para a selecção de uma determinada Meta-heurística, e especificação automática dos respectivos parâmetros, na 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. Objectivos e principais contribuições

O objectivo principal desta tese passou pelo desenvolvimento de um módulo de Auto-Optimização integrado num Sistema Multi-Agente autónomo de resolução de problemas de Escalonamento *Job-Shop* Alargado Dinâmico – o sistema AutoDynAgents – de modo a que seja utilizada uma abordagem de aprendizagem com base na experiência, para a resolução de novos problemas. Assim, as principais contribuições deste trabalho são:

- Uma revisão do estado da arte das abordagens de resolução de problemas de Optimização Combinatória, principalmente do problema do Escalonamento;
- Uma revisão do estado da arte da aprendizagem em Sistemas Multi-Agente, recaindo no problema do Escalonamento, com descrição dos aspectos dos Sistemas de Fabrico que necessitam de aprendizagem, e com a descrição de várias técnicas de aprendizagem referidas na literatura;
- A proposta de um módulo de Auto-Optimização para integração no sistema AutoDynAgents assente em Raciocínio baseado em Casos para a resolução de novos problemas de Escalonamento, com o mínimo de intervenção humana;
- Um estudo computacional do sistema AutoDynAgents com o módulo de Auto-Optimização incluído, comparando-se resultados com dados obtidos anteriormente e com um outro sistema presente na literatura.

1.3. Organização do documento

Nesta secção será descrito resumidamente cada um dos 7 capítulos que compõem esta dissertação.

Neste **primeiro capítulo** efectua-se o enquadramento desta tese, sendo apontados os principais objectivos, bem como as principais contribuições.

No **segundo capítulo** é abordado o problema do Escalonamento e são descritas algumas abordagens para a sua resolução, com especial destaque para as Meta-heurísticas, que são as técnicas usadas no âmbito deste trabalho.

No **terceiro capítulo** descreve-se os Sistemas Multi-Agente, com conceitos e definições, sendo descritos os principais modelos usados e sendo efectuado um enquadramento no escalonamento dinâmico, com descrição das principais arquitecturas usadas. É também descrita a Computação

Autónoma, suas ideias e conceitos, uma vez que é importante para o entendimento do sistema AutoDynAgents e do módulo de Auto-Optimização proposto.

No **quarto capítulo** é efectuada uma revisão do estado da arte de aprendizagem em Sistemas Multi-Agente para Escalonamento. Começa-se por descrever sucintamente a Aprendizagem Automática e suas subáreas, passando-se para a descrição do Raciocínio baseado em Casos, que apresenta uma grande importância no desenvolvimento do módulo de Auto-Optimização, e terminando com uma abordagem às questões de aprendizagem em Sistemas Multi-Agente, através do enquadramento no Escalonamento, e com a descrição das várias técnicas de aprendizagem presentes na literatura.

No **quinto capítulo** é descrito o sistema AutoDynAgents, com todos os elementos que o constituem, e o módulo de Auto-Optimização desenvolvido nesta tese, com uma descrição detalhada de todas as fases da arquitectura, sendo também ilustrado com um exemplo para um melhor entendimento do funcionamento do mesmo.

No **sexto capítulo** descreve-se o estudo computacional do sistema AutoDynAgents com inclusão do módulo de Auto-Optimização desenvolvido, retirando-se algumas conclusões sobre o mesmo.

As conclusões deste trabalho são apresentadas no **sétimo** e último **capítulo**, onde são apontadas as principais limitações, sendo apresentadas direcções para trabalho futuro relacionado com o trabalho desenvolvido.

No final encontra-se a lista de referências bibliográficas utilizadas no desenvolvimento deste trabalho.

CAPÍTULO 2. O PROBLEMA DO ESCALONAMENTO E AS ABORDAGENS PARA SUA RESOLUÇÃO

2.1. Introdução

O problema do Escalonamento pode ser definido 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" (Portmann, 1997), e têm sido desenvolvidas várias abordagens para o resolver. No entanto, muitas dessas abordagens são muitas vezes impraticáveis em ambientes de fabrico reais, intrinsecamente dinâmicos onde existem restrições complexas e uma variedade de perturbações inesperadas. Na maioria dos ambientes de mundo-real, o escalonamento é um processo reactivo progressivo onde a presença de informação em tempo real obriga continuamente à reconsideração e revisão dos planos pré-estabelecidos. A investigação em escalonamento não tem considerado muito este problema, focando-se antes na optimização dos planos de escalonamento estático (Ouelhadj & Petrovic, 2008).

Neste capítulo serão apresentados aspectos teóricos acerca de problemas de **Optimização Combinatória**, com especial realce no problema do **Escalonamento**. Sobre este problema em particular, será descrito o Escalonamento *Job-Shop*, bem como as restrições adicionais particulares da extensão *Job-Shop* Alargado (Madureira, 2003). De seguida, serão descritas algumas abordagens de resolução de problemas de Optimização Combinatória. Estas abordagens passam maioritariamente pela aplicação de métodos de aproximação, sendo estes divididos em construtivos (**Regras de Prioridade**), por melhoramentos, e compostos (**Meta-heurísticas**). É dado mais ênfase a este último tipo, as Meta-heurísticas, nomeadamente a Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, Optimização por Colónia de Formigas, e *Particle Swarm Optimization*, uma vez que foram estes os métodos usados no âmbito deste trabalho.

2.2. Optimização Combinatória

Os problemas de Optimização Combinatória surgem quando se torna necessário seleccionar, de um conjunto de dados discreto e finito, o melhor subconjunto que satisfaz determinados critérios de natureza económica-operacional.

O grande desafio da Optimização Combinatória é produzir, em tempo competitivo, soluções o mais próximo possível da solução óptima, ou seja, soluções quase-óptimas. A dificuldade surge quando, mesmo para problemas de dimensões reduzidas, 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 objectivo de percorrer todas as cidades uma única vez e voltar à cidade de partida, minimizando-se a distância total percorrida. Assim, as soluções admissíveis são dadas pela fórmula *(N-1)!,* o que para 5, 10, e 25 cidades obter-se-iam, respectivamente, 24, 362880, e 6.2×10^{23} percursos possíveis. Perante isto, rapidamente se percebe que analisar todas as soluções admissíveis não é viável.

Osman e Kelly (1996) definiram Optimização Combinatória como "o estudo matemático para encontrar uma combinação, agrupamento, ordenação ou selecção óptima de objectos discretos usualmente finito em números".

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

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

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 objectivo que atribui um custo real a cada solução $x \in C$, isto é, $f: C \to R$. 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 óptima 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 Optimização Combinatória são normalmente fáceis de descrever mas difíceis de resolver".

2.3. Escalonamento *Job-Shop*

Num problema *Job-Shop* existe um conjunto de tarefas a processar num conjunto de máquinas, sendo que cada tarefa tem uma ordem de processamento nas máquinas, isto é, cada tarefa é composta por uma lista ordenada de operações, cada uma caracterizada pela máquina onde será processada e pelo tempo de processamento respectivo (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 zero;
- Cada tarefa requer um número de operações, cada uma a ser processada numa máquina diferente;
- Os tempos de preparação das operações são independentes da sequência escalonada e estão incluídos no tempo de processamento;

- As máquinas são todas diferentes e estão continuamente disponíveis;
- As operações individuais não podem ser interrompidas, isto é, 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.

Os problemas considerados neste trabalho contêm algumas restrições adicionais ao *Job-Shop*, sendo importantes para uma representação mais real do processo de fabrico. Estas restrições são incluídas numa subcategoria de escalonamento *Job-Shop*, tendo a designação de *Job-Shop Alargado*, descrito de seguida.

2.3.1. *Job-Shop* Alargado

Os problemas *Job-Shop* Alargado consistem em problemas *Job-Shop*, com restrições adicionais, de modo a representar melhor a realidade (Madureira, 2003). Assim, estes problemas consideram outras restrições, nomeadamente:

- A existência de datas de lançamento e de entrega diferentes 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;

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

2.4. Abordagens de Resolução

Os métodos para a resolução de problemas de optimização combinatória de difícil resolução, isto é, cujo tempo de resolução aumenta exponencialmente à medida que a dimensão dos problemas aumenta, podem ser divididos em duas categorias abrangentes (Madureira, 2003): os métodos exactos e os métodos de aproximação.

Nos **métodos exactos** é realizado um processo de pesquisa exaustiva do espaço de soluções, e têm uma fundamentação na Matemática. A vantagem é que garantem a solução óptima. No entanto, devido à sua natureza, só devem ser considerados para resolução de problemas de pequena dimensão. Além disso, apresentam outras desvantagens tais como uma modelação mais complexa, podem gastar tempos proibitivos na geração da solução óptima , e nem sempre conseguem produzir uma boa solução viável em tempo considerado útil.

Os **métodos** ou algoritmos **de aproximação** têm a finalidade de encontrar soluções satisfatórias num período de tempo aceitável, e têm a sua fundamentação na Inteligência Artificial. Como vantagens refere-se a sua fácil implementação e o facto de produzirem boas soluções em tempo útil. Como desvantagem identifica-se o facto de não garantirem a obtenção da solução óptima.

Os métodos de aproximação podem ser divididos em construtivos, por melhoramentos, e compostos. Os construtivos constroem uma solução, passo a passo, segundo um conjunto de regras pré-estabelecido, como por exemplo as **Regras de Prioridade** (subsecção 2.4.1). Os métodos de aproximação por melhoramentos partem de uma solução admissível qualquer e procuram melhorá-la através de sucessivas pequenas alterações, como por exemplo o algoritmo de Pesquisa Local. Os métodos compostos têm primeiro uma fase construtiva e depois uma fase de melhoramentos, onde se podem incluir as **Meta-heurísticas** (subsecção 2.4.2).

2.4.1. Regras de Prioridade

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

As Regras de Prioridade podem ser classificadas em duas categorias, segundo a variabilidade da informação (Baker, 1974)(Madureira, 2003)(Sule, 1997):

- **Estáticas:** são decididas *a priori* e aplicadas no início do período, para períodos pré-definidos. Pressupõem que as tarefas a programar chegam em simultâneo e que esse

conjunto não varia até o processamento estar concluído. Por exemplo, as regras baseadas em datas de entrega ou em datas de lançamento;

- **Dinâmicas:** são usadas em tempo real, à medida que os eventos ocorrem, e reflectem melhor a realidade do que as regras estáticas. Por exemplo, o facto das prioridades de processamento das tarefas poderem ser alteradas em cada máquina.

Além desta classificação, Sule (1997) também diferenciou as Regras de Prioridade a partir do tipo de informação que consideram. Assim, estas podem ser **locais** se consideram apenas a informação relativa à fila de espera actual, ou **globais** se consideram dados referentes a todo o sistema de fabrico.

De seguida são apresentadas algumas das Regras de Prioridade mais utilizadas na prática (Baker, 1974)(Madureira, 2003):

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

Além destas, Madureira (2003) propôs a regra **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, e a regra **SeqNivel**, 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".

Como já referido, as regras de prioridade representam métodos de aproximação construtivos de fácil implementação e que podem encontrar soluções razoáveis facilmente. No entanto, a sua principal desvantagem recai sobre o facto de as soluções serem muitas vezes pobres devido à sua natureza míope (Ouelhadj & Petrovic, 2008).

2.4.2. Meta-heurísticas

A adaptação de ideias de várias áreas, principalmente com base na natureza, resultou no desenvolvimento das **Meta-heurísticas**, que consistem em métodos heurísticos que visam solucionar problemas complexos genéricos de Optimização Combinatória. Estas técnicas têm o objectivo de

guiar e melhorar a pesquisa de modo a superar as soluções óptimas locais, que, como já referido, constituem uma limitação no algoritmo de Pesquisa Local, e obter soluções com melhor qualidade, bastante próximas da solução óptima.

Sendo assim, as Meta-heurísticas consistem em métodos iterativos ou recursivos com o objectivo de obter soluções o mais próximo possível do óptimo global para um determinado problema. Assumindo que todas as soluções estão relacionadas entre si, podendo a partir de cada uma se alcançar o conjunto de soluções para o problema, ao caminho percorrido pelo método até encontrar a solução chama-se de **trajectória** no espaço das soluções.

Mediante um problema deve ser escolhida uma determinada Meta-heurística, sendo que essa escolha é considerada difícil, devendo para isso ser feito 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 muito do problema em causa e cada uma apresenta as suas vantagens e desvantagens, sendo que a escolha da Meta-heurística deve ter por base os resultados que se querem obter, podendo ser eficiência, eficácia, qualidade, etc.

Podem ser encontradas na literatura várias abordagens relativamente à abrangência das Meta-heurísticas, mas genericamente pode dizer-se que são constituídas por Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, Redes Neuronais, Colónia de Formigas, Algoritmos Meméticos, GRASP, entre outros (Madureira, 2003).

Seguidamente é feita uma descrição de algumas Meta-heurísticas, com especial destaque para aquelas usadas no âmbito deste trabalho, nomeadamente a Pesquisa Tabu, os Algoritmos Genéticos, o *Simulated Annealing*, a Optimização por Colónia de Formigas e o *Particle Swarm Optimization*.

2.4.2.1. Pesquisa Tabu

A **Pesquisa Tabu** foi introduzida por Glover (1986) e consiste numa estratégia de Pesquisa Local com o objectivo principal de escapar aos mínimos locais. Apresenta semelhanças 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.

Assim, a Pesquisa Tabu utiliza o conceito de memória, que é carregada com conhecimento relativo a uma aprendizagem progressiva referente ao espaço de soluções do problema. Desta forma, a ideia principal deste método é de que é possível abandonar óptimos locais consentindo uma deterioração no valor da qualidade de uma solução, desde que essa nova solução não tenha sido já visitada anteriormente. É possível ao algoritmo determinar se essa solução foi já visitada através da utilização de uma lista tabu que armazena a informação referente às soluções visitadas. Assim esta pesquisa das melhores soluções possíveis é feita através de deslocações, as quais são realizadas para as melhores soluções encontradas e seguindo certas regras. Algumas deslocações são

classificadas como proibidas, ou tabu, o que previne o aparecimento de ciclos, mínimos locais, e possibilita encontrar óptimos globais. Estas classificações são efectuadas com base na memória que serve de apoio ao algoritmo.

Esta Meta-heurística, como descrito na Figura 1, parte de uma solução inicial e escolhe para solução seguinte a melhor solução da vizinhança actual, ou duma subvizinhança, no caso de a vizinhança ser demasiado extensa para poder ser eficientemente explorada pelo algoritmo. De notar que a escolha da nova solução acontece mesmo que esta seja pior do que a solução actual, o que é uma diferença em relação ao algoritmo de Pesquisa Local.

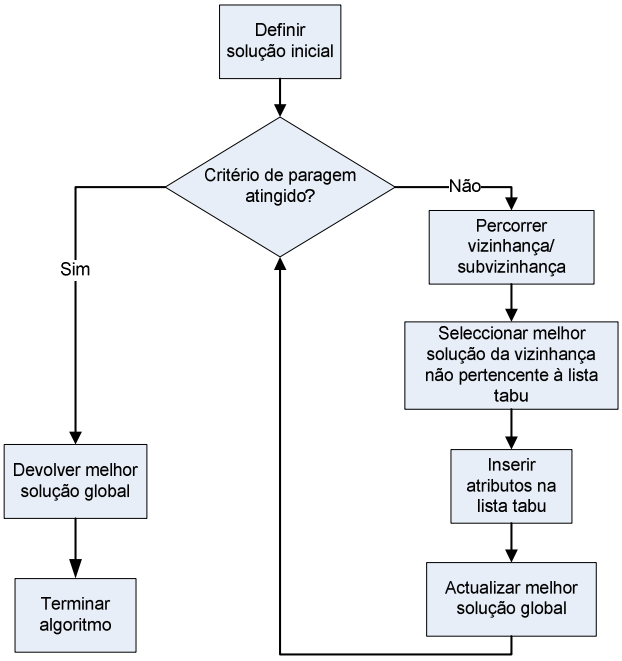

Figura 1 – Pesquisa Tabu

Tal como Madureira (2003) identificou, com esta diferença há o perigo do algoritmo entrar em ciclo, se as vizinhanças da solução actual e da nova solução forem simétricas. O problema encontra-se se for seleccionada uma nova solução, posteriormente ser seleccionada a solução anterior uma vez que essa solução se encontra presente na vizinhança da nova solução, e vice-versa, não sendo permitida a evolução do algoritmo noutro sentido, oscilando-se entre a nova solução e a solução actual.

É aqui que entra o conceito de lista tabu, que consiste numa lista com o objectivo de guardar os pares de soluções envolvidas nos últimos *k* movimentos. Se um par de soluções estiver na lista, 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 com dimensão fixa consoante são visitadas novas soluções, e uma vez atingido o limite da lista, 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 na melhoria da

função objectivo nas últimas *N* iterações, ou num número total pré-definido de iterações, momento em que o algoritmo termina.

Para que um algoritmo de Pesquisa Tabu possa ser colocado em execução é necessário ter em conta algumas configurações. É necessário 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. Para a construção dessa subvizinhança existem vários métodos, um dos quais a selecção aleatória de uma certa quantidade de soluções vizinhas da totalidade da vizinhança.

Outro aspecto importante é a definição do conteúdo da lista tabu. Como já referido anteriormente, esta lista representa a memória do algoritmo e nela são guardadas informações referentes aos movimentos anteriores efectuados na trajectória até se chegar à solução actual. Esta informação guardada pode acabar por se tornar demasiado restritiva, uma vez que se pode excluir mais soluções do que aquelas que se pretendem, mesmo soluções que ainda não foram sequer visitadas (Madureira, 2003). Para contornar esse problema, em vez de 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 actual na nova solução.

A dimensão da lista tabu corresponde ao número de registos tabu que a lista poderá conter quando o algoritmo se encontrar em execução. Não existe um valor genérico ideal para todos os problemas, o valor deve ser determinado mediante o problema em questão através de testes computacionais.

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

2.4.2.2. Simulated Annealing

O ***Simulated Annealing*** é um algoritmo relativamente antigo, com início nos anos 80 através de Kirkpatrick et al. (1983) e Cerny (1985), e com ligação à termodinâmica e à metalurgia (Metropolis et al., 1953), cuja motivação original surgiu com base no processo em que o metal em fusão é arrefecido lentamente, com tendência para solidificar numa estrutura de energia mínima. Tem uma base estatística e baseia-se em movimentos resultantes de soluções de má qualidade para uma solução actual de forma a escapar do mínimo local. É a combinação de uma geração aleatória e um melhoramento iterativo a partir de comparações e probabilidades.

No início do processo iterativo, quase todos os movimentos são aceites, isto é, são aceites todas as substituições da solução actual por uma nova solução escolhida aleatoriamente da sua vizinhança, o que permite explorar largamente o espaço das soluções. Conforme o algoritmo vai evoluindo, a temperatura vai sendo diminuída gradualmente (arrefecimento), tornando o algoritmo

progressivamente mais selectivo na aceitação de novas soluções, até que, no final, apenas são aceites os movimentos que melhoram a solução actual.

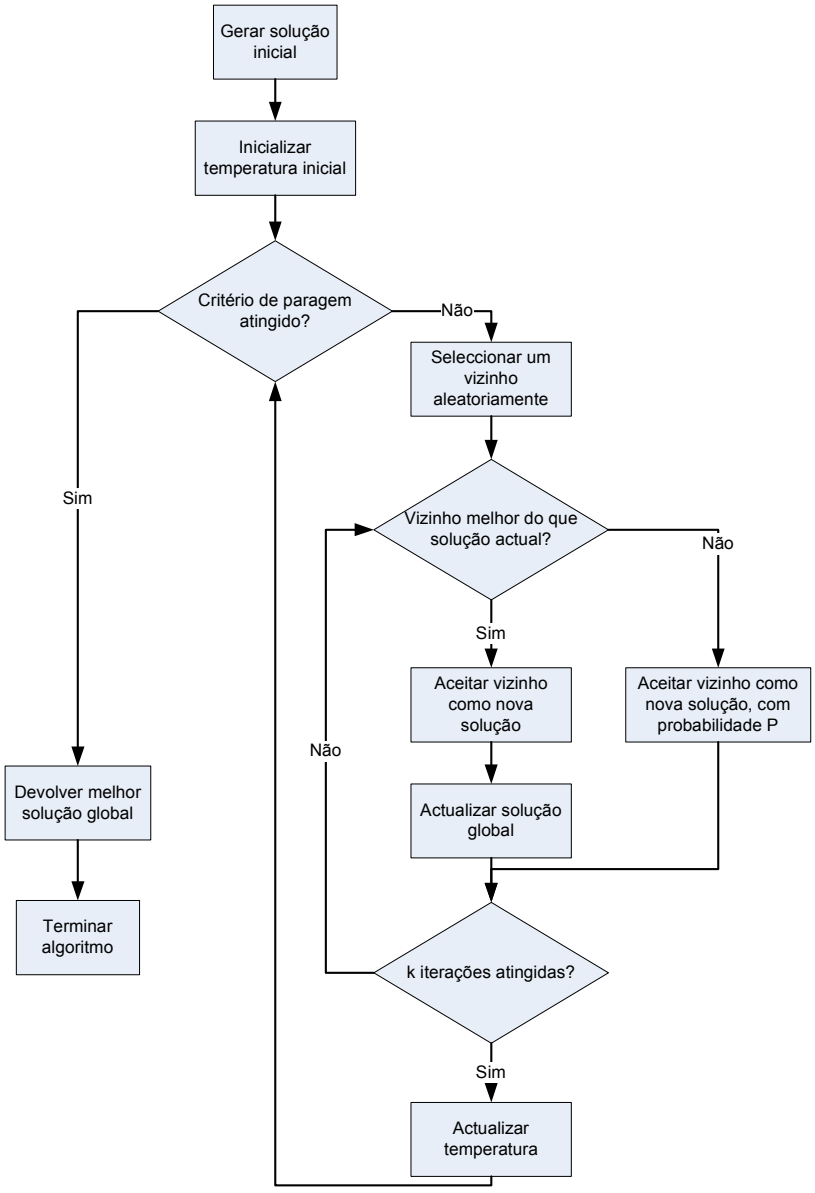

Figura 2 – *Simulated Annealing*

O algoritmo encontra-se descrito na Figura 2 e começa por gerar uma solução inicial *S*, que pode ser gerada aleatoriamente ou construída através duma heurística, e inicializa a temperatura *T* com um valor. Enquanto o critério de paragem não for atingido, para cada iteração é seleccionada aleatoriamente uma solução *S'* pertencente à vizinhança de *S*. Esta solução vizinha *S'* é avaliada para ser aceite como nova solução, o que acontece se for melhor do que *S*. Neste caso, a melhor solução global também é actualizada se necessário. Caso *S'* seja pior do que *S*, então é aceite com base numa probabilidade, geralmente com base na distribuição de Boltzmann (Blum & Roli, 2003):

$$P = \exp\left(-\frac{f(S') - f(S)}{T}\right) \quad (1)$$

Na expressão (1), *f(S')* e *f(S)* representam o valor objectivo de *S'* e *S*, respectivamente.

A escolha da temperatura, isto é, o "arrefecimento" adequado, é essencial para o desempenho do algoritmo, sendo normalmente efectuado de *k* em *k* iterações. Normalmente é aplicado um factor de redução de temperatura (α) de modo a reduzir a temperatura de forma constante, como ilustrado na equação (2), onde *k* representa o número da iteração.

$$T_{k+1} = \alpha T_k \quad (2)$$

Este processo é análogo ao processo de “cristalização” de metais e vidros, que quando arrefecido com a temperatura correcta apresenta um gasto de energia menor, de uma maneira muito sucinta a temperatura *T* diminui durante o processo de pesquisa (Blum & Roli, 2003).

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

2.4.2.3. Algoritmos Genéticos

Nos princípios da década de 70, John Holland em conjunto com os seus alunos e colegas desenvolveram pesquisas e estudos baseados na área da selecção natural de espécies, tendo alcançado um modelo formal designado por **Algoritmos Genéticos** (Holland, 1975). Estes estudos tiveram fundamento em genética e na ideia presente num livro escrito por Charles Darwin (1859) que falava sobre a teoria da origem das espécies. Neste livro, Darwin fala numa competição entre indivíduos de cada espécie animal e refere-se acerca da evolução das espécies, e que apenas os mais aptos conseguem prevalecer, sendo a natureza responsável por efectuar esta selecção natural.

Nos anos 80, David Goldberg, aluno de Holland, consegue a primeira aplicação bem sucedida destes algoritmos (Goldberg, 1989). Desde então, os Algoritmos Genéticos têm sido aplicados com sucesso nos mais diversos problemas de optimização (Madureira, 2003).

Os Algoritmos Genéticos podem ser separados em três fases fundamentais: selecção, reprodução, e substituição (Blum & Roli, 2003) (Bodenhofer, 2004) (Madureira, 2003). Na **selecção** é criada uma população temporária com alguns dos elementos da população principal. Esses elementos da população principal são os elementos mais aptos, da mesma forma que acontece no mecanismo de selecção natural onde os mais fortes prevalecem sobre os mais fracos. A esses indivíduos da população temporária são aplicados os operadores de **reprodução** (cruzamento e mutação) para ser possível criar novos indivíduos, e esses vão **substituir** os indivíduos menos aptos

da população principal. Este processo é repetido até se atingir um determinado critério de paragem, sendo que à medida que se repete, a população vai ficando com indivíduos de melhor aptidão.

A principal diferença entre estes algoritmos e outras Meta-heurísticas, tais como Pesquisa Tabu e *Simulated Annealing,* é o facto de os Algoritmos Genéticos lidarem com conjuntos de soluções (populações), em vez de lidarem com uma solução isolada. A exploração das soluções não é efectuada apenas à vizinhança de uma solução isolada, sendo pelo contrário efectuada uma exploração da vizinhança de uma população inteira. Para essa exploração e para a geração de vizinhanças das populações são usados três operadores: selecção, cruzamento, e mutação.

Uma particularidade dos Algoritmos Genéticos é o facto dos seus operadores genéticos não actuarem directamente no espaço de soluções. Em vez disso, as soluções devem ser codificadas como palavras, de comprimento e alfabeto finito (quando se refere solução, refere-se a representação codificada da solução).

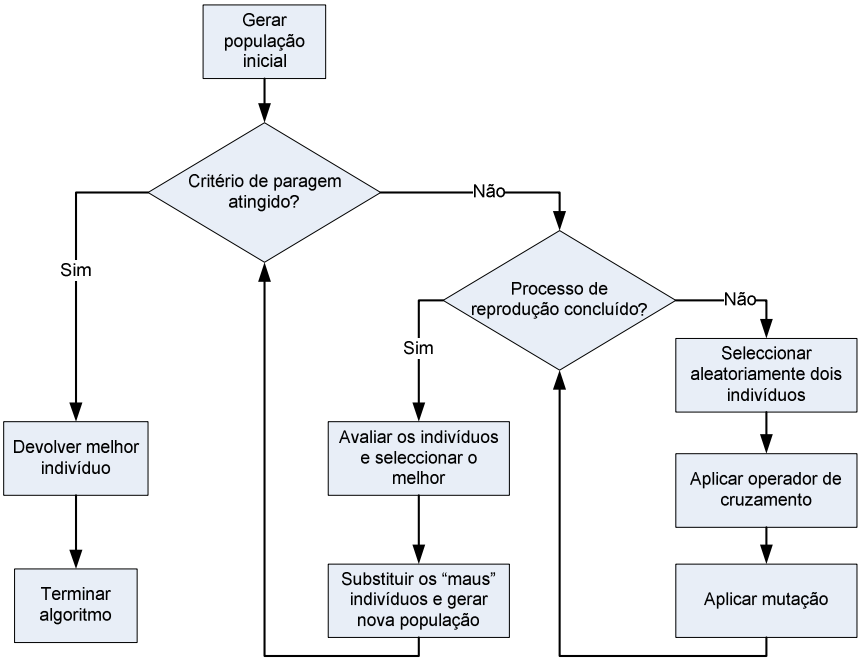

Figura 3 – Algoritmo Genético

Um Algoritmo Genético (Figura 3) 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, são seleccionados e cruzados entre si os melhores indivíduos da população, originando novos indivíduos que substituem os piores indivíduos da população actual. Cada indivíduo é avaliado pelo seu enquadramento ou aptidão, podendo este valor corresponder somente ao valor da função objectivo tanto num problema de maximização como de minimização (Madureira, 2003).

Após serem escolhidos os pares de indivíduos, 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 outra taxa (taxa de mutação), também definida previamente. Assim, estes descendentes mutados, ou não, vão substituir os indivíduos da população que se apresentem menos aptos. Este processo dá origem a uma nova população, designada por nova geração.

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 uma melhoria na aptidão dos indivíduos criados.

O **operador de selecção** tem um papel fundamental nos Algoritmos Genéticos, dado que através dele são seleccionados os indivíduos mais aptos para reprodução. Existem várias formas de efectuar uma selecção, tendo em conta vários aspectos importantes. Um desses aspectos é a selecção ser efectuada baseada em elitismo. Essa escolha pode parecer a mais óbvia, em analogia com o mundo real, onde os mais fortes prevalecem sobre os mais fracos. Assim, poderão ser descartados os mais fracos, escolhendo-se os mais aptos para a população seguinte, e para reprodução. No entanto, no caso de resolução de problemas de optimização o mesmo não acontece, pois podem ser descartados indivíduos necessários para se chegar ao melhor indivíduo possível, através de cruzamentos e mutações. Assim, os operadores de selecção mais utilizados são: selecção por elitismo, método da roleta, método da selecção universal estocástica, e método de selecção por torneios (Madureira, 2003).

Outro dos principais operadores de reprodução dos Algoritmos Genéticos é o **operador de cruzamento**, que, como já referido, tem a função de criar novos indivíduos a partir de outros seleccionados para o efeito (progenitores ou ascendentes). Estes novos indivíduos 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 (Madureira, 2003).

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 (Madureira, 2003). Em (Davis, 1991) é descrito o cruzamento *order crossover.*

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 diversas 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 óptimos locais, e consequentemente poderem ser exploradas zonas mais amplas desse espaço. Os operadores de mutação mais conhecidos em problemas de escalonamento são (Madureira, 2003): o *swap*, que consiste trocar dois genes seleccionados aleatoriamente, e o *shift*, que permite deslocar um gene relativamente à sua posição, para a direita ou para a esquerda.

Os Algoritmos Genéticos mostraram não ser muito adequados para problemas de escalonamento pois não são eficientes em encontrar uma solução quase-óptima 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 & Petrovic, 2008).

Aplicações de Algoritmos Genéticos a resolução de problemas de escalonamento podem ser encontradas em (Branke, 2000) (Fang et al., 1993) (Lee et al., 1997) (Madureira, 2003) (Madureira et al., 2007) (Madureira et al., 2009a).

2.4.2.4. Optimização por Colónia de Formigas

A Meta-heurística **Optimização por Colónia de Formigas**, *Ant Colony Optimization* em inglês, baseia-se no comportamento real das formigas, ou seja, num comportamento que permite encontrar o menor caminho entre uma fonte de comida e a respectiva colónia (Blum & Roli, 2003)(Madureira, 2003)(Maniezzo et al., 2004). Este fenómeno ocorre porque, durante a sua trajectória, as formigas depositam no caminho uma substância chamada feromona e, ao escolherem um caminho, optam, com maior probabilidade, por aquele que possui a maior quantidade de feromona, pois provavelmente será o mais curto, isto é, a trajectória 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. A feromona evapora-se ao longo do tempo e quanto maior for o percurso, maior será o impacto desta evaporação. Com o passar do tempo, as formigas seguem os caminhos com maior quantidade de feromona, e, no limite, será apenas um caminho, que corresponde ao caminho mais curto.

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

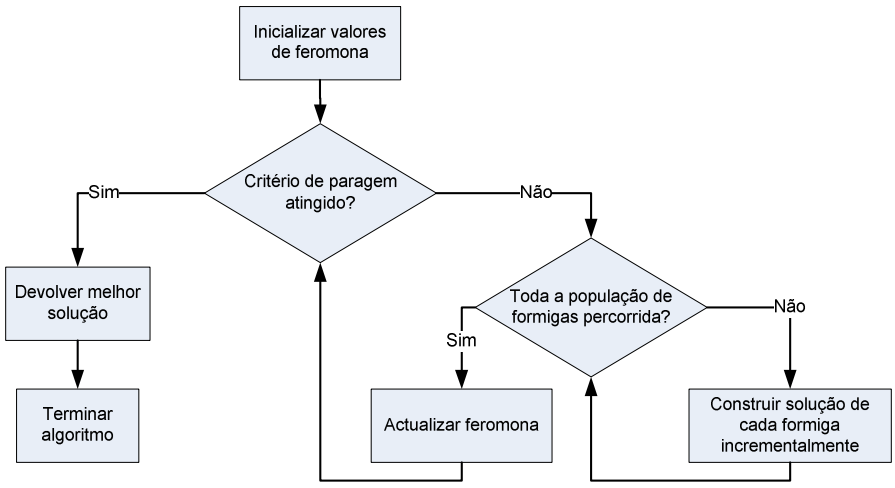

Figura 4 – Optimização por Colónia de Formigas

O algoritmo encontra-se ilustrado na Figura 4 e 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 presente. Posteriormente, quando toda a população se formigas foi tratada, a feromona é actualizada com base nestas soluções previamente construídas.

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 expressão (3), onde α e β são parâmetros para ajustar a importância da informação heurística (η_r) e os valores de feromona (τ_r), respectivamente. $J(s_a[c_l])$ refere-se ao conjunto de componentes da solução $s_a[c_l]$ onde c_l é o último componente adicionado (Blum & Roli, 2003)(Maniezzo et al., 2004).

$$\mathbf{p}(C_r|s_a[c_l]) = \begin{cases} \text{Se } c_r \in J(s_a[c_l]): \dfrac{[\eta_r]^\alpha [\tau_r]^\beta}{\sum_{c_u \in J(s_a)[c_l]} [\eta_r]^\alpha [\tau_r]^\beta} \\ \text{Senão: } 0 \end{cases} \quad (3)$$

Na fase da actualização da feromona, quando todas as formigas já construíram uma solução, é aplicada uma regra definida pela fórmula (4), onde *F* é geralmente denominada por função de qualidade, definida pela qualidade das soluções (Blum & Roli, 2003) (Maniezzo et al., 2004).

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

Esta regra para a actualização da feromona visa aumentar o valor de feromona dos componentes que encontraram as melhores soluções. De salientar que a primeira parte da expressão ($(1-\rho) * \tau_j$) 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 Optimização por Colónia de Formigas foi aplicada na resolução de problemas de escalonamento em (Haipeng et al., 2004) (Ventresca & Ombuki, 2004) (Yoshikawa & Terai, 2006)(Zwaan et al., 1999).

2.4.2.5. Particle Swarm Optimization

O ***Particle Swarm Optimization*** é uma técnica evolucionária baseada em populações, desenvolvida por James Kennedy e Russell Eberhart (1995), que pretende simular um sistema social simplificado.

Inicialmente, a ideia base seria demonstrar o comportamento que bandos de pássaros ou cardumes de peixes assumem nas suas trajectórias locais aleatórias, mas globalmente determinadas. Os bandos de pássaros ou cardumes de peixes efectuam movimentos coordenados e sincronizados como forma de descobrir comida ou como mecanismo de auto-defesa (Madureira, 2003).

Duma forma computacional, estes algoritmos surgem como uma abstracção desse comportamento biológico natural onde a procura por uma melhor posição é a busca por uma solução óptima, sendo o conjunto de posições da partícula o espaço de busca ou espaço de soluções possíveis. O comportamento de cada partícula baseia-se na sua experiência anterior e na das outras partículas com que se relaciona (Madureira, 2003). Assim, como acontece em Algoritmos Genéticos onde os indivíduos mais aptos são preservados, esta Meta-heurística salvaguarda também as melhores posições encontradas, que teoricamente significam a solução com melhor qualidade encontrada.

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

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

O algoritmo (Figura 5) começa pela inicialização da posição actual e velocidade de todas as partículas. Seguidamente, 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 actualizado o melhor valor local de cada uma (*pBest*). Depois de todas as partículas estarem tratadas, é actualizado o melhor valor global (*gBest*) e 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 actuais 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 actual de cada partícula é actualizada com base na expressão (5).

$$\mathrm{X_{id}} = \mathrm{X_{id}} + \mathrm{V_{id}} \quad (5)$$

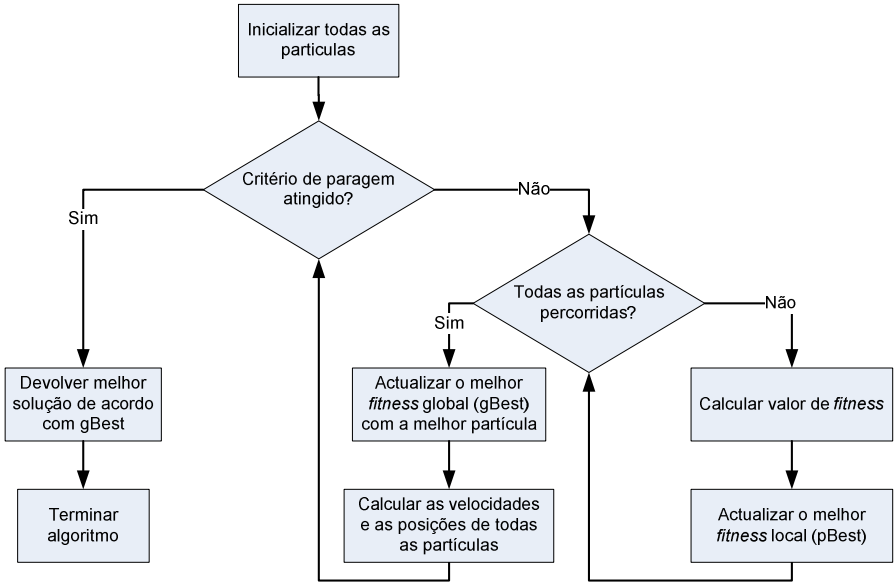

Figura 5 – *Particle Swarm Optimization*

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

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

2.4.2.6. Outras Meta-heurísticas

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 & Resende, 1989). Começa por construir uma solução inicial, utilizando uma heurística "gulosa" (*greedy*), depois aplica um algoritmo de Pesquisa Local para pesquisar a vizinhança dessa solução, guardando a melhor solução obtida. Estes dois passos são repetidos até que os critérios de paragem, normalmente número máximo de iterações, serem verificados.

A **Pesquisa Local Aleatorizada** (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 objectivo é ultrapassar as limitações da Pesquisa Local, evitando alguns óptimos locais.

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.

Os **Sistemas Imunitários Artificiais** (Dasgupta, 1999) são sistemas adaptativos inspirados no Sistema Imunológico. Um problema com solução desconhecida é tratado como antigénio enquanto potenciais soluções são modeladas como anticorpos.

A ***Harmony Search*** (Geem, 2000) é uma das mais recentes Meta-heurísticas e 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 optimização consiste em quatro passos: inicialização do problema de optimização e dos parâmetros do algoritmo, inicialização da memória *Harmony*, improvisação de uma nova harmonia a partir da procura de *Harmony*, actualização da procura de *Harmony.* No final repetem-se os últimos dois passos até ser atingido o critério de paragem.

O termo **Híper-heurística** foi introduzido por Cowling et al. (2000) e refere-se a um conjunto de algoritmos heurísticos que operam a alto nível de generalidade. São descritas como: “uma Híper-heurística gere o processo de selecção de qual a heurística ou método de nível mais baixo a ser aplicado num dado momento, dependendo das características das heurísticas e do espaço de soluções que está a ser explorado”. O domínio de uma hiper-heurística é composto por conjuntos de heurísticas (e não pelo conjunto das instâncias de um problema de optimização). Uma hiper-heurística encontra soluções indirectamente, utilizando de maneira inteligente as heurísticas que lhe foram fornecidas.

2.4.3. Outras técnicas

Alguns sistemas de resolução de problemas de escalonamento adoptaram técnicas de Inteligência Artificial tais como Sistemas baseados em Conhecimento, Redes Neuronais, Raciocínio baseado em Casos, Lógica Difusa, Sistemas Multi-Agente, etc. (Ouelhadj & Petrovic, 2008).

A motivação básica das abordagens baseadas em conhecimento é que existe uma ampla variedade de conhecimento técnico especializado nas acções correctivas a tomar na presença de eventos de tempo-real. Os Sistemas baseados em Conhecimento focam-se em capturar o conhecimento especializado dos peritos num determinado domínio e usam um mecanismo de inferência para derivar conclusões ou recomendações em relação às acções correctivas a tomar.

Possuem o potencial para automatizar o raciocínio perito humano e o conhecimento heurístico para executar sistemas de escalonamento de produção. No entanto, falta-lhes normalmente a capacidade para optimizar o sistema e requerem de esforço considerável para construir e manter. Têm o objectivo de gerar soluções exequíveis de acordo com o domínio do problema, mas, em termos de eficácia da capacidade de tomada de decisão, os Sistemas baseados em Conhecimento são limitados pela qualidade e integridade do conhecimento específico de domínio (Ouelhadj & Petrovic, 2008).

As **Redes Neuronais** (Osman & Kelly, 1996) são dispositivos inspirados no funcionamento do cérebro humano, em especial, na sua capacidade de processamento de informação em paralelo. O sistema nervoso central recebe do exterior, armazena, processa e transmite informação ao exterior. Têm a inspiração biológica no neurónio, em que existe várias entradas e uma saída. Esta saída pode estar ou não activa. Os sinais de entrada determinam se a saída fica ou não activa. As entradas são atenuadas ou amplificadas por sinapses. As redes neuronais não podem garantir decisões óptimas, mas a sua capacidade de aprendizagem torna-as ideais para sistemas de mudanças rápidas (Ouelhadj & Petrovic, 2008).

A **Lógica Difusa** (*Fuzzy Logic*) foi proposta por Zadeh (1965) com o objectivo de tratar dados vagos ou imprecisos. Assim, Zadeh introduziu o conceito de conjunto *fuzzy*, de forma a tratar matematicamente valores linguísticos imprecisos, como por exemplo “alto” ou “baixo”. Na teoria de conjuntos clássica, iniciada por Aristóteles (filósofo grego, 384 – 322 A.C.), um elemento pertence, ou não, a um dado conjunto, cujo valor lógico é 1 (verdadeiro) ou 0 (falso). Na teoria de conjuntos da Lógica Difusa, a pertença ou não de um elemento a um conjunto é indicada em vários graus, com valores compreendidos entre 0 (absolutamente falso) e 1 (absolutamente verdadeiro). Com a Lógica Difusa pretende-se então reconhecer que afirmações lógicas acerca da realidade não são absolutamente verdadeiras ou falsas, mas que têm um certo grau de verdade. Como exemplo, considere-se um copo parcialmente cheio de água, com um grau de verdade de 0.70, não estando totalmente cheio (senão o grau de verdade seria 1). A negação da afirmação consiste em dizer que o copo está vazio, com um grau de verdade de 0.30 (1 – 0.70 = 0.30). O objectivo principal da Lógica Difusa é então encontrar a “melhor” solução na presença de informação incompleta, imprecisa ou limites vagos.

O **Raciocínio baseado em Casos** (Aamodt & Plaza, 1994)(Kolodner, 1993) é uma técnica de Inteligência Artificial baseada em conhecimento que resolve problemas a partir do 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. Esta é uma das principais técnicas usadas no desenvolvimento deste trabalho, pelo que uma descrição mais detalhada pode ser encontrada na secção 4.3.

Os **Sistemas Multi-Agente** são sistemas nos quais existe um conjunto de agentes que interagem entre si e tentam, cooperativa ou competitivamente, chegar a uma solução para resolver o problema em questão. Têm vindo a ser cada vez mais usados para a resolução dos mais variados

tipos de problemas, entre eles os problemas de escalonamento. Uma vez que o âmbito deste trabalho se inclui nos Sistemas Multi-Agente, pode ser encontrada uma descrição mais alargada na secção 3.2.

2.5. Sumário

Neste capítulo foram descritos os problemas de Optimização Combinatória, e, uma vez que o âmbito deste trabalho recai sobre o problema do Escalonamento, foram descritas as restrições presentes no escalonamento *Job-Shop* e na sua extensão *Job-Shop* Alargado, importante para uma representação mais rigorosa de sistemas reais de produção.

Foram abordadas algumas estratégias de resolução de problemas de Optimização Combinatória, onde se inclui o problema de Escalonamento, sendo descritos os métodos de aproximação construtivos (onde se inclui as Regras de Prioridade), por melhoramentos, e compostos (Meta-heurísticas). De acordo com o âmbito deste trabalho, foi feita uma descrição mais detalhada acerca das Meta-heurísticas usadas, nomeadamente a Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, Optimização por Colónia de Formigas, e *Particle Swarm Optimization*, com alguns exemplos de aplicação ao problema de Escalonamento.

CAPÍTULO 3. OS SISTEMAS MULTI-AGENTE E A COMPUTAÇÃO AUTÓNOMA

3.1. Introdução

Nos últimos anos, tem havido um interesse crescente em abordagens descentralizadas para a resolução de problemas complexos de mundo real, como o problema do Escalonamento, e são cada vez mais as soluções propostas e as implementações bem sucedidas. Muitas dessas abordagens recaem na área dos Sistemas Distribuídos, onde um número de entidades trabalha em conjunto para, cooperativamente, resolver problemas. Nesta área salientam-se os Sistemas Multi-Agente, que se preocupam com a coordenação dos comportamentos de um conjunto de agentes de modo a partilhar conhecimento, capacidades, e objectivos tendo em vista a resolução de problemas complexos. Devido ao crescimento exponencial da complexidade dos sistemas, é importante que estes sejam cada vez mais autónomos de modo a responder a dinamismo, sobrecargas e a recuperarem de falhas.

Neste capítulo será feita uma descrição do conceito de **agente** e de **sistemas baseados em agentes**. Será também descrito o problema do **Escalonamento Dinâmico** em Sistemas Multi-Agente, com destaque para as arquitecturas usadas. Por último, e uma vez que foi importante o seu estudo para o desenvolvimento deste trabalho, será apresentada a **Computação Autónoma**.

3.2. Sistemas Multi-Agente

Antes dos anos 80, a comunidade científica em Inteligência Artificial preocupava-se em desenvolver um sistema único que concentrava as capacidades inteligentes dos sistemas, o chamado Sistema Inteligente. A partir daí, e após vários encontros e debates para trocas de ideias, surgia uma nova área de actuação, resultado da junção entre a Inteligência Artificial e a Computação Distribuída, sendo designada de **Inteligência Artificial Distribuída**.

Davis (1980) referiu que o objectivo da Inteligência Artificial Distribuída é a resolução de problemas de situações que não são possíveis de solucionar com apenas uma entidade munida de inteligência artificial. Por seu lado, Nilson (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.

Tradicionalmente, a Inteligência Artificial Distribuída está dividida em duas áreas (Panait & Luke, 2005). 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 colectiva de uma

solução para o problema. A segunda classe de abordagens, os Sistemas Multi-Agente, destaca os comportamentos comuns dos agentes com algum grau de autonomia e a complexidade que surge das suas interacções. Estes sistemas preocupam-se com a coordenação dos comportamentos de uma determinada comunidade de agentes, de modo a partilhar conhecimento, capacidades, e objectivos tendo em vista a resolução de problemas complexos.

Para se usar correctamente um Sistema Multi-Agente 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 às arquitecturas multi-agente. Assim, nesta secção serão descritos os conceitos de "agente" e de "Sistema Multi-Agente", com alguns exemplos de aplicação.

3.2.1. 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.

Seguidamente serão descritas algumas definições do termo "agente" bem como as suas características e limitações.

3.2.1.1. Definições de agente

De acordo com os dicionários da Porto Editora[1], o termo "agente" apresenta, entre outras, as seguintes definições:

1. Que actua;
2. Pessoa encarregada de praticar certas operações materiais ou representar os interesses de outrem;
3. Aquilo que age, produz um efeito; o que origina (algo);

Do ponto de vista informático, a definição do termo "agente" não é consensual, sendo no entanto possível encontrar na literatura algumas definições descritas na Tabela 1.

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 interacção entre os agentes, entre outras, todas elas descritas mais pormenorizadamente na próxima subsecção.

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

Tabela 1 – 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 acções com sentido num mundo real."
(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 subrotina, 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."
(Coelho, 1994)	"Os agentes, para sobreviver, são forçados a possuir capacidades de tomada de decisão, estratégica e previsional, de coordenar as suas acções entre si e de enfrentar tarefas complicadas de forma efectiva."
(Russell & Norvig, 1995)	"Um agente é algo que sente o ambiente onde existe através de sensores e actua através de actuadores."
(Maes, 1995)	"Os agentes habitam ambientes dinâmicos e complexos, sentem e agem autonomamente nesses ambientes e realizam um conjunto de objectivos ou tarefas para as quais foram concebidos."
(Wooldridge & Jennings, 1995)	"Os agentes são sistemas baseados em *hardware* ou *software* que têm as seguintes propriedades: autonomia (operar sem intervenção directa de humanos e controlar as suas próprias acções), capacidade social (interagir com outros agentes, ou humanos), reacção (os agentes têm percepção sobre o ambiente, físico ou não, e respondem atempadamente às alterações que ocorrem), e pro-actividade (também são capazes de exibir comportamentos guiados por objectivos, tomam a iniciativa)."
(Franklin & Graesser, 1996)	"Um agente autónomo é um sistema que está contido e faz parte de um ambiente e actua sobre esse ambiente ao longo do tempo, possuindo a sua própria agenda."
(Panait & Luke, 2005)	"Um agente é um mecanismo computacional que exibe um alto grau de autonomia, executa acções no seu ambiente baseado em informação (sensores, *feedback*) recebida a partir do ambiente."
(Bernon et al., 2005)	"Um agente é uma entidade autónoma com uma dimensão reactiva, proactiva, e social."

3.2.1.2. Características e tipos de 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 objectivos, através de sensores e actuadores. Assim, um agente pode apresentar algumas ou mesmo todas as propriedades descritas na Tabela 2.

Tabela 2 – Propriedades dos agentes (Ramos & Silva, 2008)

Propriedade	Descrição
Capacidade sensorial	Capacidade de possuir sensores para captar informação relativa ao ambiente envolvente. Estes sensores podem ser físicos ou virtuais, dependendo do tipo de agente.
Reactividade	Permite que o agente possa sentir e reagir mediante as alterações que vão ocorrendo no seu meio. Um agente puramente reactivo actua de um modo sensação-reacção, isto é, os sensores despoletam uma acção que já estava predefinida internamente para aquela sensação. Existe a vantagem de os agentes responderem rapidamente a mudanças no meio. Como desvantagens, não é possível ao agente considerar acções alternativas e cada solução para cada problema tem de ser codificada com antecedência. Para resolver estes problemas, é possível que os agentes reactivos sejam monitorizados e controlados ou até mesmo modificáveis, sendo possível introduzir novos comportamentos reactivos. Contrariamente ao agente reactivo existe o agente deliberativo em que se mantém uma representação interna do ambiente envolvente, que pode ser modificado através de raciocínio. Na prática, um agente nunca é puramente reactivo nem puramente deliberativo. Scheutz e Schermerhorn (2003) efectuaram uma comparação entre tipos de agente, tendo em conta a sua reactividade, chegando à conclusão de que os agentes deliberativos com capacidades de previsão obtiveram os melhores resultados.
Autonomia	Permite que o agente decida e controle as suas próprias acções. A autonomia é uma característica marcante e completamente aceite dos agentes, tal como Franklin e Gasser (1996) referiram. A autonomia é fundamental em situações onde a intervenção do ser humano é dificultada ou em situações onde o ser humano pode tomar decisões incorrectas. Um agente puramente autónomo não bloqueia nem pára quando outros agentes ou seres humanos não são capazes de cumprir uma dada tarefa. No entanto também se admite a existência de agentes semi-autónomos, que dependem de outros agentes ou de seres humanos, onde estes normalmente cumprem diferentes funções ou validam as decisões do agente.
Pro-actividade	Possibilita que o agente se oriente através de determinados objectivos e não actue apenas para responder ao ambiente. Um agente pró-activo tem uma visão própria, guiada por objectivos, e é um agente interveniente capaz de alterar o rumo do ambiente no qual opera.
Persistência	Um agente persistente é um agente que existe ao longo do tempo. Normalmente os agentes têm um ciclo normal (criação-execução-extinção) mas há casos em que os agentes devem ter uma duração muito mais ampla, tal como agentes que representem máquinas de uma fábrica, agentes que representem vendedores em comércio electrónico, agentes que efectuam o processamento de alarmes, entre outros.
Capacidade social	Permite que os agentes comuniquem com outros agentes e talvez com seres humanos, cooperando, concorrendo ou competindo para a obtenção de soluções. Na cooperação os agentes comunicam entre si com a finalidade de atingirem os seus objectivos ou obterem algum benefício. A partilha é um aspecto determinante para os agentes cooperantes. Na competição os agentes devem possuir características mais desenvolvidas na

	observação do ambiente em que actuam, bem como serem capazes de perceber melhor os seus concorrentes.
Aprendizagem	Permite que os agentes mudem as suas acções mediante experiências anteriores. 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.
Mobilidade	Um agente móvel é capaz de se movimentar de máquina para máquina, de modo a satisfazer os seus objectivos. Um agente com esta característica pode, num determinado momento, abandonar o ambiente computacional onde operava e passar para outro ambiente computacional distinto. O agente escolhe quando e para onde migrar, e pode suspender a sua execução numa máquina a partir do ponto em que foi suspenso.
Flexibilidade	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.
Agilidade	A agilidade é uma característica que se relaciona com a capacidade que um agente tem para responder rapidamente a novas oportunidades que apareçam, mesmo que tal implique executar algo que não estava previsto anteriormente.
Carácter	O carácter ou personalidade de um agente permite que este tenha comportamento emocional de modo a que seja credível.
Inteligência	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 acções, corrigir os erros e reagir a situações não esperadas, sendo capaz de se adaptar e aprender.

A classificação de um agente é feita com base nas características apresentadas. Não é fácil encontrar um agente que possua todas as características mencionadas, embora, tal como já referido, algumas sejam fundamentais para o estabelecimento do conceito de agente.

3.2.1.3. Limitações

O uso de agentes nem sempre é vantajoso, sendo possível identificar limitações em algumas situações (Ramos & Silva, 2008):

- Quando se usam sistemas baseados em agentes **não é possível manter um controlo total do sistema**. As soluções baseadas em agentes podem não ser adequadas para problemas onde restrições globais tenham de ser mantidas, onde seja necessário um comportamento de tempo-real e onde devam ser evitados *deadlocks*;
- **É impossível ter uma perspectiva global** pois a acção de um agente é determinada pelo seu estado local, logo o conhecimento global completo não é possível devido aos agentes poderem apenas atingir globalmente soluções sub-óptimas;

- A já referida característica da credibilidade leva à **falta de confiança** na delegação de tarefas aos agentes. As organizações têm pouca experiência com este tipo de sistemas, e isso converge num problema de confiança na utilização de agentes. Essa confiança tem que ser ganha por parte destes sistemas, e isso demora o seu tempo.

As duas primeiras limitações descritas advêm, de forma directa, do facto dos sistemas baseados em agentes serem distribuídos. No entanto, a terceira é a que mais condiciona o uso de soluções baseadas em agentes, pois é difícil confiar em agentes de modo a permitir que estes substituam por completo os seres humanos. Assim, os sistemas baseados em agentes devem ser vistos como Sistemas de Apoio à Decisão, de modo a que os utilizadores possam aceitar, modificar ou rejeitar as soluções propostas pelos agentes.

Estes problemas poderão ser resolvidos com mais alguma investigação e desenvolvimento e, por exemplo, com a criação de agentes que possuam características muito próprias que lhes permitam colmatar essas falhas, quando inseridos num determinado ambiente.

3.2.2. Sistemas baseados em Agentes

Os sistemas baseados em agentes podem ser divididos em dois tipos diferentes (Stone & Veloso, 1997): os **Sistemas** centralizados **de Agente Único** e os **Sistemas Multi-Agente**. 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 objectivos, acções 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 Multi-Agente diferem dos Sistemas de Agente Único pelo facto de existirem vários agentes que modelam os objectivos e acções dos outros agentes. Num cenário multi-agente típico, existem interacções 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 Multi-Agente são considerados dinâmicos, isto é, o ambiente pode ser alterado enquanto um determinado agente actua.

Panait e Luke (2005) definiram um ambiente multi-agente 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 Multi-Agente. De outra forma, os agentes distribuídos podem agir sincronizados, sabendo exactamente 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 interacções, 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 respectivos. 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 interacções 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 onde os outros agentes escolheram para explorar, eles podem ser codificados para operar identicamente a uma configuração mestre-súbdito.

Os Sistemas Multi-Agente têm vindo a destacar-se nas duas últimas décadas como um paradigma particularmente interessante para a modelação e implementação de algumas aplicações em ambientes imprevisíveis e dinâmicos (Zambonelli & Parunak, 2004). Luck et al. (2005) referem mesmo que estes sistemas podem ser o próximo grande passo na evolução da computação, tal como a Orientação por Objectos o foi.

A utilização de agentes deve-se a uma visão holística, onde o total é mais do que a soma das partes. Os Sistemas Multi-Agente 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 sub-problemas 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.

Nesta secção serão descritas as arquitecturas de agentes e Sistemas Multi-Agente, bem como os modelos existentes destes últimos. Serão também apresentadas algumas áreas de aplicação.

3.2.2.1. Modelos de Sistemas Multi-Agente

Em (Horling & Lesser, 2004) os autores descreveram vários modelos de Sistemas Multi-Agente, descritos de seguida.

Os **modelos hierárquicos** são os mais utilizados no desenho de Sistemas Multi-Agente. 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 interacções 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 susceptibilidade a *bottlenecks* se a gestão da informação não for bem concretizada.

Os **modelos holónicos** representam sistemas em que cada sistema é constituído por sub-sistemas 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 objectivo comum, e deixam de existir quando esse objectivo 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 actividades de modo a que o objectivo 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 objectivo 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 actuem em prol do objectivo 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 interacção 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 objectivo 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 objectivos (estimulo à congregação). No entanto é necessário que ambos os agentes retirem benefícios substanciais da

congregação pois há um elevado tempo e energia dispendida 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 objectivos 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 promova a obtenção dos objectivos 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 *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 interacções, 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 objectivos 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** são simplesmente combinações dos modelos acima descritos, e possuem as vantagens e desvantagens dos tipos de modelos adoptados.

Na Tabela 3 é apresentado um resumo das características, vantagens e desvantagens dos diferentes modelos apresentados nesta subsecção.

Tabela 3 – Características, vantagens e desvantagens dos modelos de Sistemas Multi-Agente (Horling & Lesser, 2004)

Modelo	Característica principal	Vantagens	Desvantagens
Hierárquico	Decomposição	Mapeia vários domínios comuns	Frágeis; podem levar a *bottlenecks*
Holónico	Decomposição com autonomia	Explora a autonomia de unidades funcionais	Deve organizar os *holons*; falta de previsão de desempenho
Aliança	Dinâmico, dirigido por objectivos	Explora força nos números	Os benefícios de curto prazo podem não superar os custos de construção da organização
Equipa	Coesão a nível de grupo	Lida com problemas de grande granularidade; Centrado nas tarefas	Aumento da comunicação entre agentes
Congregação	Vida longa, dirigido por utilidade	Facilita a descoberta de agentes	Os conjuntos podem ser excessivamente restritivos
Sociedade	Sistema aberto	Serviços públicos; convenções bem definidas	Potencialmente complexos, os agentes podem requerer capacidades adicionais relacionadas com sociedade
Federação	Agentes intermédios	Facilita a alocação dinâmica de agentes	Os intermediários podem originar *bottlenecks*
Mercado	Competição	Bom na alocação; utilidade acrescida através da centralização; equidade acrescida através de licitações	Potencial para colusão, comportamentos maliciosos; a complexidade da decisão da alocação pode ser elevada
Matricial	Gestores múltiplos	Partilha de recursos	Potencial para conflitos; necessária uma elevada sofisticação dos agentes
Composto	Organizações concorrentes	Explora os benefícios de vários modelos	Sofisticação ampliada; desvantagens de vários modelos

3.2.2.2. Áreas de Aplicação

Os Sistemas Multi-agente têm vindo a ser bastante utilizados nas mais diversas áreas, com um maior destaque no Comércio Electrónico, Robótica, Controlo de Tráfego e Sistemas de Produção e de Escalonamento. Seguidamente serão descritas resumidamente cada uma destas áreas de aplicação.

A convergência da computação e das telecomunicações tem levado a que a actividade comercial seja bastante diferente do que era há alguns anos atrás, principalmente depois da generalização da Internet, cujo crescimento ultrapassou as previsões dos mais optimistas. Uma grande parte dos

utilizadores da Internet já experimentaram comprar algo online. A introdução de agentes no **Comércio Electrónico** faz todo o sentido, na medida em que os agentes podem automatizar um conjunto de funções de um modo muito mais rápido e automático do que os seres humanos. Isto é particularmente importante em mercados nos quais o preço dos produtos esteja em constante mutação. Os agentes tanto podem ser centrados nos compradores como nos vendedores, sendo que os primeiros serão responsáveis por procurar e adquirir produtos com base na personalização imposta pelo utilizador, enquanto os segundos serão responsáveis por divulgar os produtos e garantir a segurança das transacções. Assim, existem várias questões importantes no uso de agentes nesta área de aplicação, tal como a segurança e a confiança dos agentes. Exemplos de trabalhos de aplicação de agentes em Comércio Electrónico podem ser encontrados em (Link-Pezet et al., 2000) (Lin & Kuo, 2002) (Viamonte et al., 2006) (Jascanu, 2008).

O uso de Sistemas Multi-Agente na **Robótica** faz todo o sentido, uma vez que os robôs podem ser representados por agentes, com os sensores e os actuadores. Dois exemplos práticos são a exploração colaborativa, em que vários agentes dotados de autonomia e inteligência decidem as trajectórias 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 reactivos, e havendo planeamento estratégico e planeamento táctico. Alguns trabalhos nesta área são (Burgard et al., 2005) (Toriz et al., 2009) (Stone, 2000) (Kose et al., 2005) (Bezek et al., 2006) (Rostami et al., 2005).

O **Controlo de Tráfego** é uma aplicação crítica caracterizada pela complexidade, mas com uma natureza distribuída. Pode representar tráfego aéreo, em que existem agentes para representar voos, pistas de aterragem, factores 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. Exemplos de trabalhos recentes nesta área são (Gorodetsky et al., 2008) (Kuyer et al., 2008) (Shi & Huang, 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 Multi-Agente têm sido bastante usados nesta área e tem havido realce para a componente de negociação. Existem várias implementações em **Escalonamento**. De acordo com Shen e Norrie (1999), têm sido tradicionalmente implementadas três tipos de arquitecturas: modelos hierárquicos, federação e autónomos. Se assumirmos que modelos holónicos e organizações matriciais pertencem à classe de arquitecturas hierárquicas, então já se alarga o número de arquitecturas usadas. Alguns trabalhos com agentes na área de Sistemas de Produção e Escalonamento podem ser encontrados em (Lin & Solberg, 1994) (Maturana, 1997) (Maturana et al., 1999) (Shen et al., 2000) (Ouelhadj et al., 2000) (Shen et al., 2001) (Madureira et al., 2007) (Madureira et al., 2009a) (Madureira et al., 2009b).

3.3. Escalonamento Dinâmico em Sistemas Multi-Agente

A maioria dos sistemas de escalonamento e controlo desenvolvidos em ambientes industriais têm sido vistos tradicionalmente como um processo *top-down* de comando e de resposta que se baseia em modelos centralizados e hierárquicos (Bongaerts et al., 2000) (Parunak, 1996) (Shen & Norrie, 1999)(Shen et al., 2001). Para assegurar a consistência dos dados através de toda a empresa, os sistemas de escalonamento centralizados e hierárquicos (Figura 6) baseiam-se em bases de dados centrais. Para optimizar o desempenho, as decisões de escalonamento são feitas centralmente ao nível do supervisor, e são então distribuídas para o nível de recursos de produção para execução. Uma arquitectura comum dá a um computador central a responsabilidade do escalonamento, do despacho de recursos, da monitorização de distúrbios, e da aplicação das acções correctivas.

Os sistemas de escalonamento centralizados fornecem uma visão global consistente do estado da empresa e também bons planos de escalonamento. No entanto, a experiência prática tem indicado que estes sistemas tendem a apresentar limitações perante perturbações (Ouelhadj & Petrovic, 2008).

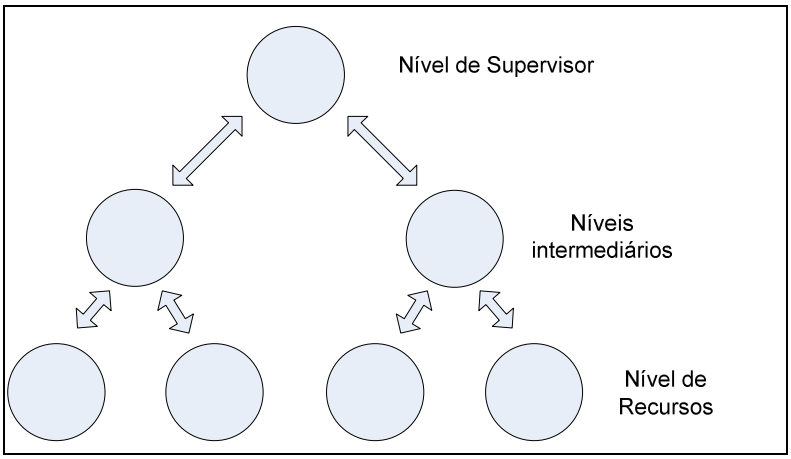

Figura 6 – Arquitectura centralizada e hierárquica

Os sistemas de escalonamento centralizados e hierárquicos apresentam também algumas desvantagens (Bongaerts et al., 2000) (Parunak, 1996) (Shen & Norrie, 1999). A principal refere-se à existência de um computador central, que pode constituir um constrangimento (*bottleneck)* e limita a capacidade da fábrica, sendo um ponto de falha único que pode levar a que toda a fábrica falhe. Além do mais, modificar a configuração dos sistemas de produção hierarquicamente controlados é dispendioso e consome muito tempo, pois envolve uma dispendiosa reescrita do *software*. Outra desvantagem é que a variação no fluxo de informação aumenta o tempo de latência do processo de tomada de decisão. Além disso, a experiência prática tem indicado que os sistemas de

escalonamento centralizados e hierárquicos tendem a ter problemas na reacção a perturbações e podem falhar a responder eficazmente à presença de eventos em tempo-real. Quando ocorre alguma perturbação, é direccionada para o nível mais alto na hierarquia, e apenas após o plano ter sido adaptado, o novo plano desencadeia um novo fluxo de comandos que formam as reacções à perturbação. Esta variação da informação resulta em lentos tempos de resposta levando a uma baixa robustez. Apesar dos sistemas de escalonamento hierárquicos e centralizados poderem fornecer planos de escalonamento globalmente melhores em ambientes onde as perturbações de tempo-real são raras, cada vez mais se constata que estes são ineficientes a responder a ambientes altamente dinâmicos.

A pressão competitiva global nos ambientes de produção tem resultado em mudanças fundamentais na operação dos sistemas de fabrico. Os sistemas de hoje em dia devem adaptar-se rapidamente às perturbações enquanto mantêm pequenos ciclos de produtos, produtividade melhorada, e flexibilidade operacional elevada. Para fazer face a este desafio, a tendência actual tem sido no sentido de sistemas altamente autónomos capazes de oferecer robustez, estabilidade, adaptabilidade, e um uso eficiente dos recursos disponíveis através de uma arquitectura distribuída e modular (Parunak, 1996) (Shen et al., 2001). Existe uma tendência crescente no sentido da definição de organizações distribuídas como resultado da necessidade para níveis de responsabilidade elevados da fábrica para mudanças nos mercados e nas tecnologias. A motivação principal de projectar estes sistemas consiste em descentralizar o controlo do sistema de fabrico, reduzindo assim a complexidade e os custos, aumentando a flexibilidade, e melhorando a tolerância a falhas.

Existe uma evidência substancial de que os Sistemas Multi-Agente constituem uma das abordagens mais promissoras na construção de sistemas de fabrico complexos, robustos, e económicos devido à sua natureza autónoma, distribuída e dinâmica, e à sua robustez contra falhas (Parunak, 1996) (Shen et al., 2001). O uso de Sistemas Multi-Agente para resolver o problema do escalonamento dinâmico é motivado pelos seguintes aspectos (Parunak, 1996) (Shen & Norrie, 1999) (Shen et al., 2001):

1. **Primeiro**, os sistemas de escalonamento baseados em agentes reconhecem que os dados e o controlo são distribuídos pela fábrica. Estes sistemas são compostos por agentes autónomos anexados a cada entidade física ou funcional na instalação (recursos, operadores, tarefas, etc.). A autonomia local permite aos agentes tomar a responsabilidade de realizar planos de escalonamento locais para uma ou mais entidades no processo de produção e responder localmente e eficientemente a variações locais, aumentando a robustez e a flexibilidade do sistema;
2. **Segundo**, estes agentes individuais têm uma liberdade considerável para responder a condições locais e interagir e cooperar entre eles de modo a atingir óptimos globais e planos de escalonamento robustos. O desempenho global do sistema não é planeado globalmente, mas emerge através das interacções dinâmicas dos agentes em tempo-real.

Deste modo, o sistema emerge a partir de decisões locais concorrentes e independentes dos agentes;

3. **Terceiro**, o *software* de cada agente é muito menor e simples do que numa abordagem centralizada, e como resultado é mais fácil de ser escrito, instalado e mantido. Além disso, é possível integrar novos recursos ou remover alguns existentes com os seus agentes anexados sem proceder a alterações de *software* existente no sistema, bastando apenas conecta-los à rede de agentes do sistema.

A principal motivação no desenho de Sistemas Multi-Agente para escalonamento dinâmico é descentralizar o controlo dos sistemas de fabrico, reduzindo assim a complexidade, aumentando a flexibilidade e a tolerância a falhas. Recusando a ideia tradicional de um sistema de escalonamento central, que estabelece um plano de fabrico para todas as tarefas e máquinas, os Sistemas Multi-Agente assumem a presença de vários agentes com uma boa autonomia na tomada de decisão, distribuídos num sistema de fabrico. Os agentes interagem e cooperam ou competem/negoceiam uns com os outros de forma a atingir desempenhos globais eficazes. A autonomia local permite aos agentes ter a responsabilidade de realizar planos de escalonamento para um ou mais componentes funcionais ou físicos no processo de produção (tal como máquinas ou tarefas). Os agentes têm a capacidade de observar o seu ambiente e de comunicar e coordenar com os outros agentes de forma a assegurar que os planos de escalonamento locais levam a planos globais desejáveis. A autonomia local permite aos agentes responder localmente às perturbações locais, aumentando a robustez e a flexibilidade do sistema. A tecnologia multi-agente aparece como uma abordagem adequada para o escalonamento dinâmico pois apresenta capacidades tais descentralização, integração, robustez, e flexibilidade.

Vários estudos comparativos têm discutido as características dos Sistemas Multi-Agente que os tornam candidatos atractivos para implementar escalonamento dinâmico em contraste com os sistemas de escalonamento centralizados e hierárquicos. Parunak (1996) demonstrou que os Sistemas Multi-Agente estão bem adaptados para aplicações modulares, descentralizadas, susceptíveis a frequentes mudanças, mal estruturadas, e complexas. Shen e Norrie (1999) discutiram as vantagens do uso dos Sistemas Multi-Agente em escalonamento de produção identificando capacidades de integração, robustez e reactividade, flexibilidade, heterogeneidade, e autonomia.

3.3.1. Arquitecturas para Escalonamento baseadas em Agentes

Um crescente número de empresas está a recorrer a tecnologia baseada em agentes para abordar os ambientes complexos e dinâmicos comuns à maioria das empresas e têm sido atingidos resultados bem sucedidos. Ouelhadj e Petrovic (2008) referiram duas arquitecturas usadas na maioria das aplicações, as **arquitecturas autónomas**, onde vários agentes conseguem autonomamente definir os seus planos de escalonamento, reagir a mudanças e cooperar entre eles, e as

arquitecturas com mediador, onde existem agentes mediadores para coordenar o comportamento dos restantes agentes com vista à obtenção de planos de escalonamento globais.

3.3.1.1. Arquitecturas Autónomas

Nas arquitecturas autónomas (Figura 7), os agentes representam entidades de fabrico, tais como recursos e tarefas, e têm a capacidade de definir os seus planos de escalonamento locais, reagir a mudanças, e cooperar directamente entre eles para gerar o óptimo global ou um plano robusto (Ouelhadj & Petrovic, 2008). Os modelos de Sistemas Multi-Agente que têm uma arquitectura autónoma são os modelos hierárquicos, os modelos holónicos, os modelos em aliança, os modelos em equipa, os modelos em congregações, e os modelos em sociedade.

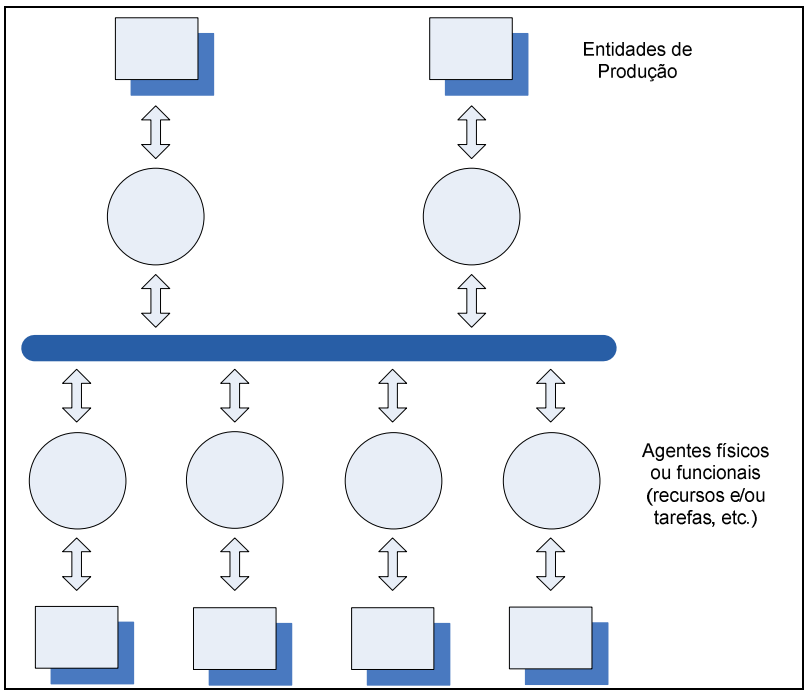

Figura 7 – Arquitectura autónoma (adaptada de (Ouelhadj & Petrovic, 2008))

Yams (***Y**et **A**nother **M**anufacturing **S**ystem*) (Parunak, 1987) é um dos primeiros sistemas de fabrico baseados em agentes que atribui um agente a cada nó na hierarquia de controlo (fábrica, célula, estação de trabalho, máquina, tarefa). A ideia principal do sistema Yams consiste nos agentes tarefa negociarem com os agentes recurso para atribuição de tarefas aos agentes máquina usando o Protocolo de Rede de Contratos (Smith, 1980).

Shaw (1988) desenvolveu um sistema de escalonamento dinâmico para um sistema de fabrico celular. Um agente célula pode subcontratar trabalho a outras células através do Protocolo de Rede de Contratos. As solicitações de mensagens de oferta são difundidas para as células, e estas avaliam as especificações das operações e submetem as ofertas, que descrevem a sua estimativa no tempo

de conclusão ou tempo de processamento mais curto. A célula que optimiza o critério predefinido é seleccionada para realizar a operação.

Ouelhadj et al. (2000) propuseram uma arquitectura multi-agente simples para escalonamento dinâmico em sistemas de fabrico flexíveis que envolve apenas agentes recurso. Os agentes recurso são responsáveis pelo escalonamento dinâmico dos recursos, não detêm controlo uns sobre os outros, e negoceiam através do protocolo rede de contratos para produzir um plano de escalonamento global. Cada agente recurso efectua as seguintes operações: escalonamento, detecção, diagnóstico e reparação de erros. Os agentes recurso reagem localmente aos eventos de tempo-real ocorridos no recurso correspondente usando as acções correctivas descritas numa base de conhecimento. Quando os eventos de tempo-real ocorrem, tais como falhas nas máquinas, o agente recurso negoceia as operações das tarefas em falha para encontrar agentes recurso alternativos.

Recentemente foram propostos contratos de compromisso nivelados como uma extensão do protocolo de rede de contratos para uma maior eficiência económica dos contratos entre agentes na presença de informação incompleta e eventos futuros (Ouelhadj & Petrovic, 2008).

Alguns sistemas de escalonamento baseados em multi-agentes usam leilões e mecanismos monetários para negociação inter-agente. Os agentes trocam recursos por dinheiro cujos preços são determinados através da comunicação de ofertas. Lin e Solberg (1994) propuseram uma arquitectura multi-agente para escalonamento dinâmico baseada num modelo monetário que combina os objectivos de escalonamento e um mecanismo de preços. O modelo proposto trata cada tarefa e recurso como um agente. Os agentes tarefa negoceiam com os agentes recurso via um mecanismo de rede de contratos de oferta para optimizar um objectivo ponderado em função da data de entrega, preço, qualidade, e outros factores definidos pelo utilizador.

Outros sistemas de escalonamento baseados em agentes usaram abordagens de aprendizagem para escalonamento dinâmico. Aydin e Öztemel (2000) propuseram um sistema de escalonamento para *Job-Shop* dinâmico com agentes, usando Aprendizagem por Reforço. O agente é treinado por um algoritmo melhorado de Aprendizagem por Reforço através da fase de aprendizagem e de seguida toma as decisões para escalonar as operações com sucesso. O sistema de escalonamento consiste em duas partes: o ambiente simulado e o agente inteligente. O agente selecciona a Regra de Prioridade mais adequada para seleccionar a tarefa para atribuir uma máquina de acordo com as condições da fábrica, enquanto o ambiente simulado realiza actividades de escalonamento usando a regra seleccionada pelo agente.

Pendharkar (1999) propôs uma abordagem de aprendizagem baseada em agentes para escalonamento dinâmico. Na arquitectura multi-agente, as áreas de trabalho são controladas por agentes com uma base de conhecimento contendo as regras de prioridade e usa aprendizagem baseada em Algoritmos Genéticos para actualizar as regras na base de conhecimento em intervalos

de tempo periódicos. Uma maior frequência de aprendizagem pode ajudar um agente a adaptar-se rapidamente a variações na fábrica.

Em (Madureira et al., 2007) é proposto um Sistema Multi-Agente para escalonamento dinâmico distribuído da produção, usando Algoritmos Genéticos e Pesquisa Tabu, com a designação de MASDScheGATS (*Multi-Agent System for Distributed Manufacturing Scheduling with Genetic Algorithms and Tabu Search*). Neste sistema, cada agente representa uma unidade de processamento, máquinas ou tarefas. O problema de escalonamento é decomposto numa série de problemas de máquina única (Madureira, 2003), sendo que cada agente máquina resolve localmente os seus problemas de escalonamento, através da aplicação de Algoritmos Genéticos ou Pesquisa Tabu, procura e encontra soluções óptimas ou quase-óptimas. Após todos os agentes máquina terem resolvido os seus problemas locais, estes são integrados segundo um mecanismo de coordenação das soluções, de modo a serem respeitadas as precedências das tarefas, para ser criado um plano de escalonamento global.

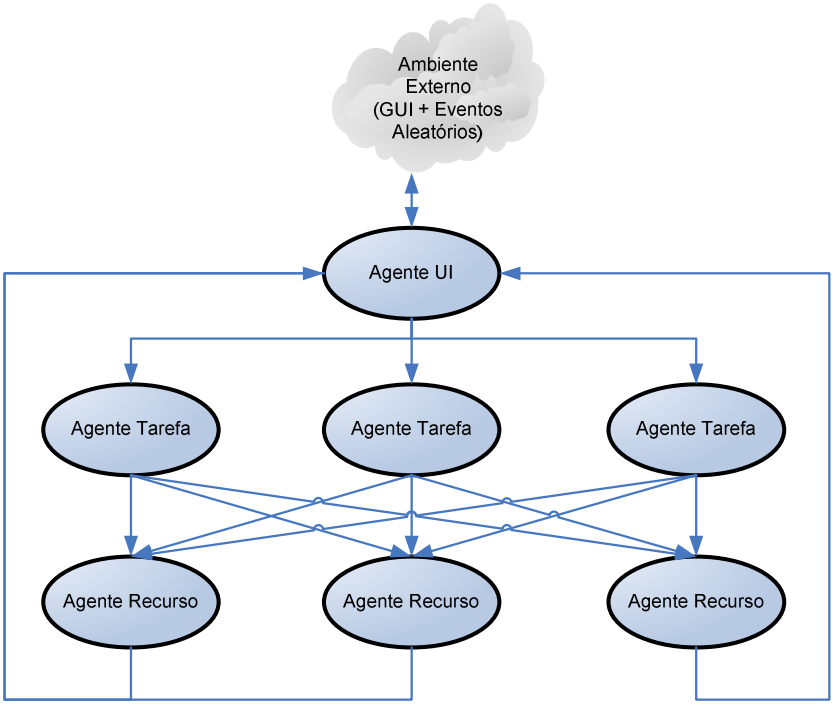

Figura 8 – Arquitectura do sistema MASDScheGATS (Madureira et al., 2007)

O sistema MASDScheGATS possui um modelo em equipa e a sua arquitectura encontra-se ilustrada na Figura 8, baseando-se em três diferentes tipos de agentes:

- **AgenteUI:** além de ser responsável pela interface de comunicação com o utilizador, o AgenteUI é também responsável pela criação dinâmica dos AgentesTarefa necessários de acordo com o número de tarefas do problema de escalonamento, por atribuir cada

tarefa ao agente respectivo, e também por verificar e coordenar as soluções recebidas pelos AgentesMáquina;

- **AgenteTarefa:** processa a informação necessária da tarefa, ou seja, é responsável pela geração dos tempos mais cedo e mais tarde de processamento e pela comunicação de cada operação ao AgenteMáquina respectivo;
- **AgenteMáquina:** é responsável por escalonar as operações recebidas dos AgentesTarefa. Este agente aplica Meta-heurísticas (Algoritmos Genéticos e Pesquisa Tabu) para encontrar as melhores soluções possíveis num tempo de processamento aceitável e comunica ao AgenteUI para este verificar as soluções e aplicar um mecanismo de coordenação de modo a ser gerado um plano de escalonamento final.

3.3.1.2. Arquitecturas com Mediador

Apesar do bom desempenho das arquitecturas autónomas, estas normalmente enfrentam problemas em fornecer planos de escalonamento optimizados globalmente na presença de um grande número de agentes, tal como em empresas virtuais (Bongaerts et al., 2000) (Shen & Norrie, 1999) (Shen et al., 2001). Vários investigadores propuseram arquitecturas com mediador para escalonamento dinâmico em ambientes complexos considerando a combinação de robustez e de planos globais óptimos e previsíveis (Ouelhadj & Petrovic, 2008).

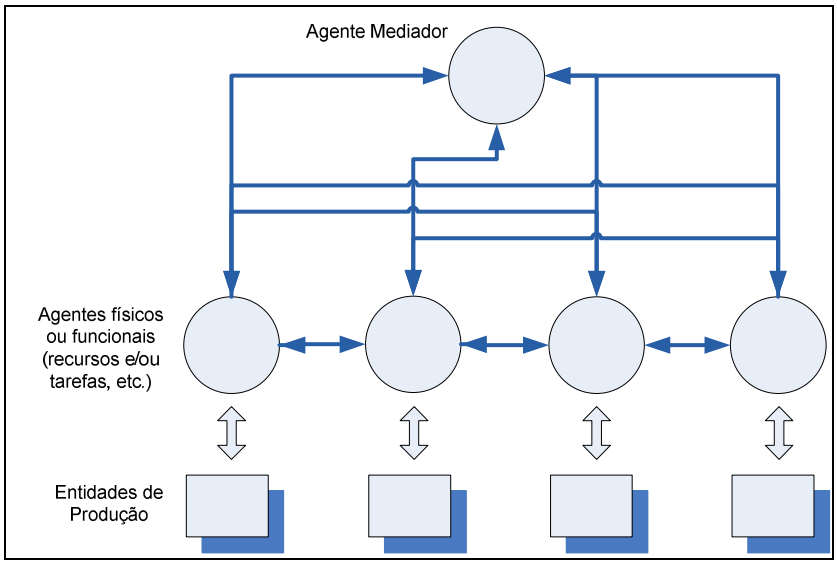

Figura 9 – Arquitectura com mediador (adaptada de (Ouelhadj & Petrovic, 2008))

Uma arquitectura com mediador tem uma estrutura básica que consiste em agentes locais autonomamente cooperantes que são capazes de negociar entre eles tendo em vista atingir alvos de produção virtuais (Bongaerts et al., 2000) (Shen & Norrie, 1999) (Shen et al., 2001). Esta estrutura básica é estendida com agentes mediadores para coordenar o comportamento dos agentes locais para realizar escalonamento dinâmico global (Figura 9). Fornece uma simplicidade computacional,

enquanto é bastante adequada para desenvolver sistemas industriais distribuídos que são complexos, dinâmicos, e compostos por um grande número de agentes recurso. Os modelos em federação, os modelos de mercado e os modelos baseados em organizações matriciais representam arquitecturas com mediador.

Os agentes mediadores operam de forma concorrente com os agentes locais e contribuem para o mesmo processo de tomada de decisão como os agentes locais. Os agentes locais mantêm a sua tomada de decisão autonomamente, mas podem pedir conselhos ao agente mediador. Este agente tem a capacidade de aconselhar, impor ou actualizar decisões tomadas pelos agentes recurso de forma a satisfazer os objectivos globais e resolver situações de conflito. Desta forma, o agente mediador pode coordenar e influenciar o comportamento global do sistema. O agente mediador tem uma visão global de todo o sistema, enquanto os agentes locais podem ter uma visão mais detalhada, mais correcta e mais actualizada das situações locais. Como tal, os agentes locais podem reagir mais rapidamente a perturbações, enquanto os agentes mediadores podem coordenar os comportamentos dos agentes e melhorar frequentemente o desempenho global.

Bongaerts et al. (2000) mostraram nos seus estudos comparativos que as arquitecturas com mediador têm um melhor desempenho relativamente às arquitecturas autónomas, devido à sua capacidade de previsão em combinação com a sua capacidade de reagir a perturbações, que podem resultar em desempenhos globalmente satisfatórios.

Uma arquitectura com mediador muito básica foi proposta por Ramos (1994) para escalonamento dinâmico em sistemas de fabrico flexíveis. A arquitectura é composta por agentes tarefa, agente administrador de tarefas, agentes recurso, e agente mediador de recursos. O agente administrador de tarefas cria os agentes tarefa. O agente mediador de recursos negoceia com os agentes recurso a execução das tarefas usando o protocolo de redes de contratos. Quando ocorrem falhas nos recursos, são enviadas mensagens sobre as operações em falha para o agente mediador de recursos, que procede com o processo de renegociação com outros agentes recurso. Este modo de lidar com as falhas é bastante simples.

Para uma melhor robustez em sistemas de fabrico complexos, alguns autores propuseram a integração de agentes mediadores em cada nível da fábrica. Maturana et al. (1999) propuseram a arquitectura com mediador no sistema Metaphor I, para escalonamento dinâmico de sistemas de fabrico grandes e heterogéneos para lidar com questões de empresas virtuais combinando subtarefas com agrupamento virtual de agentes. As questões de parceria da empresa virtual são associadas com a unificação dos subsistemas de fabrico heterogéneos numa grande coligação virtual dinâmica de subsistemas cooperativos. Existem dois tipos principais de agentes nesta arquitectura: os agentes recurso e os agentes mediadores. Os agentes recurso são usados para representar dispositivos e operações de fabrico, enquanto os agentes mediadores são usados para coordenar os agentes recurso usando o protocolo de redes de contratos. O mau funcionamento de um agente recurso é mantido a um nível local. São simulados colapsos nos recursos através da introdução de períodos de falhas no recurso. Cada tarefa alocada dentro do período morto é reescalonada para outros *slots* de

tempo disponíveis descobertos no mesmo recurso (o recurso com mau funcionamento) ou em diferentes recursos. Com base no Metaphor I, Shen et al. (2000) desenvolveram o sistema Metaphor II, para integrar as actividades de fabrico da empresa tal como projecto, planificação, escalonamento, simulação, execução, fornecimento de material, e serviços de marketing. Nesta arquitectura o sistema de fabrico é organizado através duma hierarquia de mediadores de subsistema. Foram introduzidos quatro tipos de mediadores: empresa, recurso, marketing, e projecto. Cada subsistema é um sistema baseado em agentes integrados no sistema através de um mediador especial. Os agentes recurso são coordenados por mediadores apropriados a todos os níveis do sistema. Um mediador de alto nível coordena os mediadores de baixo-nível tais como máquinas, ferramentas, trabalhadores, e mediadores de transporte. A cooperação entre os agentes recurso é realizada através da combinação do mecanismo de mediação e o protocolo de rede de contratos. Foram desenvolvidos vários mecanismos de reparação de planos de escalonamento para responder à presença de eventos de tempo-real, tais como: chegada de novas tarefas, eliminação de tarefas, falhas nas máquinas, e atrasos no tempo de processamento de tarefas.

Sun e Xue (2001) desenvolveram uma arquitectura de escalonamento reactivo com mediador para responder a mudanças nas tarefas e nos recursos. Os recursos são representados por agentes que são coordenados por dois mediadores, um mediador da instalação e um mediador do pessoal, usando o protocolo de rede de contratos. O escalonamento reactivo é conduzido para modificar o plano criado para responder a mudanças nas tarefas tais como eliminação de tarefas ou inserção de tarefas urgentes, e condições de fabrico tais como falhas nas máquinas, ou doenças súbitas nos trabalhadores durante o processo de produção. São usadas estratégias de reescalonamento e colaboração baseada em agentes para reparar apenas parte do plano originalmente criado para melhorar a eficiência do escalonamento reactivo, enquanto se mantém a qualidade de escalonamento.

3.3.1.3. Conclusões

Os estudos comparativos de Shen e Norrie (1999), Bongaerts et al. (2000), e Shen et al. (2001) relataram que as arquitecturas autónomas têm perspectivas de reduzir a complexidade, integridade, custo-eficácia, alta flexibilidade, e apresentam uma alta robustez contra perturbações. São adequados para aplicações com um pequeno número de agentes. No entanto o comportamento do sistema pode ser imprevisível em ambientes complexos com um grande número de agentes. Em contraste, as arquitecturas com mediador mostram uma melhoria do desempenho relativamente às arquitecturas autónomas para desenvolvimento de sistemas de fabrico distribuídos, complexos e compostos por um grande número de agentes, tais como empresas virtuais. Combinam robustez contra perturbações com optimização e previsibilidade do desempenho global.

3.4. Computação Autónoma

O crescimento exponencial da capacidade computacional juntamente com o aparecimento da comunicação em rede, principalmente a Internet, levou as empresas a investir significativamente em infra-estruturas e aplicações computacionais. Estes sistemas são sujeitos a falhas, dinamismo, sobrecargas, etc., devido ao crescimento exponencial da sua complexidade. Todos os dias, as empresas investem cada vez mais na manutenção do que em desenvolvimento. Perante a previsão da chegada de um "ponto de ruptura", em Outubro de 2001, Paul Horn (2001), vice-presidente da IBM Research, deu corpo e visibilidade a este problema, lançando um desafio da IBM com a designação de Computação Autónoma (*Autonomic Computing*).

A Computação Autónoma é um paradigma da computação em que grandes sistemas informáticos contêm mecanismos de manutenção quotidiana da própria aplicação embutidos, com o objectivo de automatizar essa manutenção. Assim, as aplicações devem ser capazes de se acomodar, aproveitar e proteger das mudanças no ambiente e nos seus objectivos.

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 correctamente ou não era possível a concentração noutros aspectos da vida.

Computacionalmente, pretende-se um sistema pró-activo, capaz de se auto-gerir, mediante determinadas regras, objectivos e ordens dadas pelo administrador, sem que se pretenda excluir completamente a intervenção humana do sistema.

Nesta secção são abordados os diversos componentes de auto-gestão e os elementos básicos de uma arquitectura de um Sistema Autónomo.

3.4.1. Comportamentos de Auto-Gestão

Como já referido, um Sistema Autónomo é um sistema com capacidade de auto-gestão. A finalidade destes sistemas é 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 objectivo. 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 propositadamente (Kephart & Chess, 2003).

Assim, foram definidas algumas propriedades, propostas inicialmente com o nome **auto-*** (em inglês, *self-**) que devem ser próprias de sistemas que implementem o paradigma da Computação Autónoma (Kephart & Chess, 2003) (Parashar & Hariri, 2006): **Auto-Configuração** (*Self-Configuring*), **Auto-Optimização** (*Self-Optimizing*), **Auto-Recuperação** (*Self-Healing*), e **Auto-Protecção** (*Self-Protection*). Estas propriedades serão descritas de seguida.

3.4.1.1. Auto-Configuração

Os vários processos de instalação, configuração, e integração de sistemas complexos constituem actualmente um desafio, pelo que há grandes perdas de tempo e 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 uma **Auto-Configuração** com base em políticas de alto-nível previamente indicadas. Assim, quando um novo componente é inserido no sistema, este incorpora-se de forma autónoma no sistema, tal como uma nova célula se insere no corpo humano, ou mesmo como uma pessoa se insere numa população.

Com a Auto-Configuração pretende-se portanto adaptar ou dotar o sistema de uma série de características que sejam as mais apropriadas, mediante o conhecimento existente, para se integrar e configurar dentro de um novo ambiente (Kephart & Chess, 2003).

3.4.1.2. Auto-Optimização

Com as centenas de parâmetros de configuração que podem ser alterados e que tornam os sistemas difíceis de optimizar, a **Auto-Optimização** emerge como um conceito que permite aos Sistemas Autónomos procurarem continuamente formas de melhorar a sua execução, 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 & Chess, 2003).

Permite também que um Sistema Autónomo faça a sua própria monitorização, experiência, planeamento e afinação de parâmetros, com base no conhecimento adquirido do contacto com o ambiente, para melhorar activamente o seu desempenho e efectuar as escolhas mais apropriadas para alcançar um objectivo pretendido.

3.4.1.3. Auto-Recuperação

Os Sistemas Autónomos devem ser capazes de detectar e diagnosticar as fontes do problema resultantes de falhas de *software* e *hardware* e proceder à reparação necessária para o contínuo funcionamento do sistema, tomando acções 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 objectivos pretendidos, podendo a partir dessas previsões proceder às devidas alterações.

3.4.1.4. Auto-Protecção

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-Protecção** é defender o sistema em dois sentidos:

1. **Primeiro**, deve protegê-lo a larga escala contra ataques maliciosos e intrusivos e defendê-lo contra possíveis falhas que fiquem por corrigir pela Auto-Recuperação;
2. **Segundo**, deve ser capaz de, com base em relatórios de sensores, prever problemas que possam vir a ocorrer para que estes possam ser evitados ou atenuados. Estes problemas podem ser comportamentos errados que provocam desvios ao objectivo pretendido. Mal estes problemas sejam detectados, devem ser tomadas decisões baseadas em políticas que tentam corrigir o sistema para o tornar menos vulnerável.

3.4.2. Arquitectura

Um sistema autónomo é constituído por vários elementos que colaboram entre si para lhe proporcionar todas as características de que necessita. De uma forma genérica, um sistema autónomo pode ser dividido nos seguintes componentes (Kephart & Chess, 2003) (IBM, 2005) (Parashar & Hariri, 2006), ilustrados na Figura 10:

- Pontos de Contacto (*Touchpoints*);
- Gestores Autónomos (*Autonomic Managers*);
- Recursos Geridos (*Managed Resources*);
- Fontes de Conhecimento (*Knowledge Sources*);
- Interface com o utilizador (*Manual Manager*).

Estes componentes encontram-se interligados e comunicam através de serviços de comunicação (*Enterprise Service Bus*). Seguidamente são descritos cada um destes componentes.

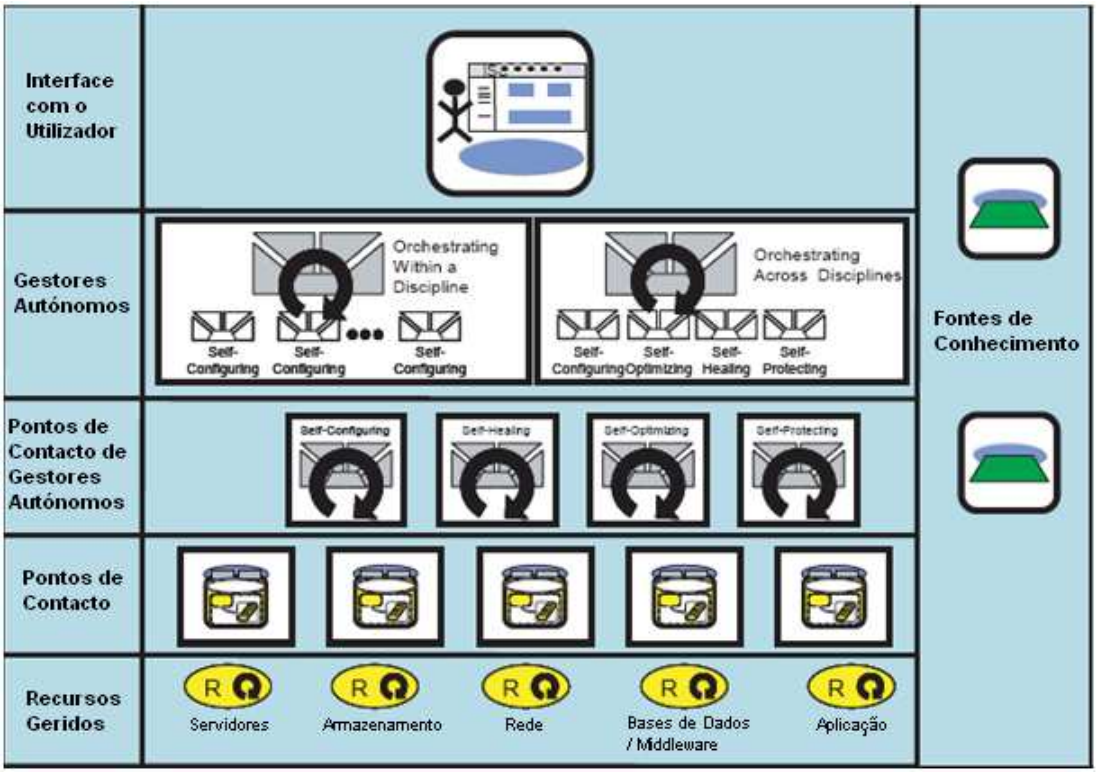

Figura 10 – Arquitectura de um sistema de Computação Autónoma (adaptado de (IBM, 2005))

3.4.2.1. Pontos de Contacto

Nos Sistemas Autónomos, os Pontos de Contacto, representam as interfaces de gestão para os Recursos Geridos. Um Ponto de Contacto activa 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 interacção e controlo dos recursos, o Sensor e o Actuador. 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 Actuador, por sua vez, representa um conjunto de operações "*set*", que de alguma maneira permitem alterar o estado de um recurso gerido.

3.4.2.2. Gestores Autónomos

Um Sistema Autónomo é composto por um ou mais Gestores Autónomos, tendo cada um a função de monitorizar o sistema continuamente 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 Actuadores a permitirem modificá-lo.

O ciclo de controlo de um Gestor Autónomo encontra-se ilustrado na Figura 11, consiste num ciclo fechado, e é composto pelas seguintes fases (Kephart & Chess, 2003):

- **Monitorização:** fornece os mecanismos para recolher, agrupar, filtrar e reportar detalhes acerca de um 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 ajudam o Gestor Autónomo a aprender sobre o sistema e a prever situações futuras semelhantes. A análise é efectuada com base na informação recolhida pela monitorização e a interpretação e avaliação com base no conhecimento do sistema;
- **Planeamento:** fornece os mecanismos necessários para construir planos de acção, mediante políticas armazenadas, que permitem remediar situações analisadas anteriormente;
- **Execução:** fornece os mecanismos que executam as acções indicadas pelo planeamento. Os planos devem ser colocados em execução com o mínimo de interferência e de conflitos entre si. As alterações são efectuadas através dos Actuadores dos Recursos Geridos que necessitem de ser alterados.

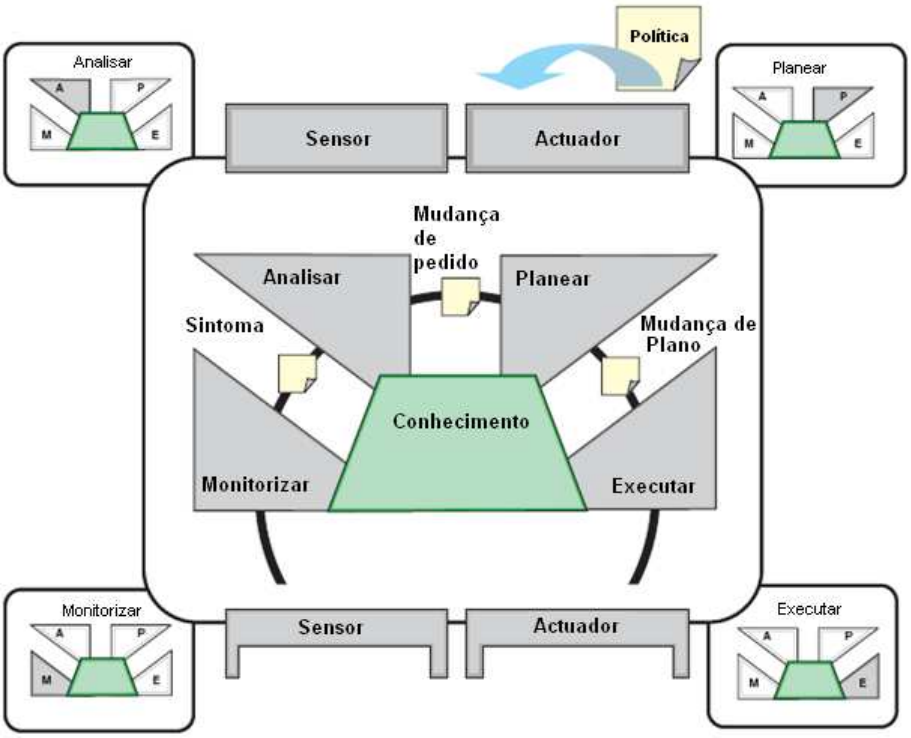

Figura 11 – Funcionamento de um Gestor Autónomo (adaptado de (IBM, 2005))

Um Gestor Autónomo possui as funções acima indicadas, contudo podem ser feitas algumas alterações que possam ser úteis. Pode ser configurado para efectuar apenas determinadas partes do

ciclo de controlo, ficando deste modo dedicado a essa finalidade. Outra configuração possível é juntar dois Gestores Autónomos que estejam configurados com funções de monitorização e análise, e planeamento e execução respectivamente, de forma a criar um ciclo fechado mais completo.

3.4.2.3. Recursos Geridos

Os Recursos Geridos englobam todo o tipo de recursos de *hardware* e *software* que os componentes do sistema controlam, e tanto podem ser servidores como aplicação, unidades de armazenamento, serviços, ou outras entidades.

Um Recurso Gerido pode conter embutido o seu próprio ciclo de controlo de auto-gestão, 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 que é fornecida para o Recurso Gerido.

3.4.2.4. Serviços de Comunicação

Os Serviços de Comunicação, ou *Enterprise Service Bus* (Sweitzer & Draper, 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;
- Activar um gestor autónomo para controlar múltiplos Pontos de Contacto;
- Activar um Gestor Autónomo para controlar um único Ponto de Contacto;
- Activar múltiplos gestores autónomos para controlarem múltiplos Pontos de Contacto.

Sweitzer e Draper (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.

3.4.2.5. Fontes de Conhecimento

Central a todas as actividades de um Sistema Autónomo está uma base de conhecimento que é substanciada com diferentes funcionalidades perante os requisitos de aplicações específicas.

As Fontes ou bases de Conhecimento são qualquer tipo de registos de informação que possibilitem armazenar conhecimento. Consistem em tipos particulares de dados arquitectados em sintaxes e semânticas assim como sintomas, políticas, alteração de planos ou pedidos, de forma a poderem ser partilhados pelos gestores autónomos do sistema (Kephart & Chess, 2003).

À medida que o sistema se vai auto-desenvolvendo, este aprende a detectar novos tipos de comportamento, e obtém 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.

Os Gestores Autónomos podem obter conhecimento de três formas diferentes:

1. **A informação é enviada**: uma política é enviada e consiste numa colecção de comportamentos limitados ou preferenciais que influenciam as decisões que o Gestor Autónomo tem que tomar;
2. **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;
3. **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 actualizar o conhecimento para registar as acções que foram efectuadas como resultado da análise e planeamento. O Gestor Autónomo pode então indicar que efeitos tiveram as acções efectuadas no Recurso Gerido, após monitoriza-lo. 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 conhecimento, armazenados segundo sintaxes e semânticas comuns de forma que possam ser partilhados por diversos componentes (Kephart & Chess, 2003):

- **Conhecimento de 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, é um tipo de conhecimento que é consultado para se determinar se é ou não necessário efectuar acções no sistema. Todos os Gestores Autónomos podem ter acesso a elas, e dessa forma todo o sistema é gerido através de uma certa colecção de políticas. O sistema deve igualmente fornecer interfaces para a criação, alteração e implantação de políticas, e mesmo saber distinguir se as regras estão especificadas correctamente, protegendo-se de erros humanos;
- **Conhecimento para detecção de problemas**: são incluídos sintomas, dados monitorizados, árvores de decisão, e conhecimento criado pelo processo de detecção de erros. Como o sistema reage de forma a corrigir problemas detectados, as informações relativas a essas correcções são armazenadas como conhecimento.

3.4.2.6. Interface com o Utilizador

A interface com o utilizador promove uma apresentação consistente aos utilizadores permitindo-lhes inserir determinados tipos de informações necessárias desde políticas, objectivos ou ordens 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 necessitarem de receber formações para lidarem com os novos componentes introduzidos no sistema.

3.5. Sumário

Neste capítulo foram descritos os Sistemas Multi-Agente, desde a descrição do conceito de agente, passando pelas suas características e limitações. Foram referidas várias arquitecturas e modelos de Sistemas Multi-Agente, terminando a secção com exemplos de áreas de aplicação.

Foi também abordado o Escalonamento Dinâmico em Sistemas Multi-Agente com descrição e comparação de arquitecturas usadas na resolução deste problema, nomeadamente as arquitecturas autónomas e as arquitecturas com mediador.

Para terminar o capítulo, foi feita uma breve abordagem à Computação Autónoma descrevendo-se os comportamentos de auto-gestão e aspectos na arquitectura dos Sistemas Autónomos.

CAPÍTULO 4. A APRENDIZAGEM EM SISTEMAS MULTI-AGENTE PARA ESCALONAMENTO

4.1. Introdução

Os Sistemas Multi-Agente actuam tipicamente em ambientes complexos, abertos, dinâmicos e imprevisíveis. Assim, a aprendizagem torna-se num factor por vezes inevitável. A aprendizagem é uma área relevante da Inteligência Artificial assim como da inteligência humana. Plaza et al. (1996) 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 muito útil, é uma visão rígida e limitada da aprendizagem. De uma forma mais geral, considera-se aprendizagem como a aquisição de novo conhecimento ou a actualização do conhecimento existente.

Neste capítulo serão abordados vários conceitos inerentes ao conceito de Aprendizagem em Sistemas Multi-Agente, com destaque aplicacional na área do Escalonamento. Em primeiro lugar será feita uma descrição resumida da **Aprendizagem Automática** com as suas subáreas. Será também descrito o Raciocínio baseado em Casos, uma vez que esta é a principal técnica usada no desenvolvimento do módulo de aprendizagem. Finalmente, será abordada a **Aprendizagem em Sistemas Multi-Agente**, aplicada ao Escalonamento, com a descrição de vários métodos e técnicas referidas na literatura.

4.2. Aprendizagem Automática

A Aprendizagem Automática é uma área de investigação da Inteligência Artificial que se preocupa com o desenvolvimento de algoritmos e técnicas para dotar os computadores com a capacidade de aprendizagem, com o mínimo de intervenção humana. Mitchell (1997) definiu-a da seguinte forma: "a área da Aprendizagem Automática preocupa-se com a questão de como construir programas computacionais que melhorem automaticamente com a experiência". Por seu lado, Alpaydin (2004) refere que "a Aprendizagem Automática é programar os computadores de forma a optimizar um critério de desempenho através de exemplos ou experiência passada". Em ambos os autores é notório o mesmo conceito que se baseia num único termo: experiência. A experiência define-se como o "acto ou efeito de experimentar", mas também como o "conhecimento obtido pela prática de uma actividade ou pela vivência", ambos os conceitos retirados dos dicionários da Porto Editora[2]. É assim possível concluir da necessidade de haver um período de existência para se poder

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

experimentar, para se poder adquirir a experiência com a qual é possível desenvolver a capacidade de aprendizagem, sendo isto válido para os humanos, animais ou mesmo computadores/máquinas.

A Aprendizagem Automática consiste então no estudo dos métodos que possibilitam aos computadores aprender. Os computadores são usados na resolução de um variadíssimo número de tarefas, cuja maioria é relativamente de fácil implementação por parte dos programadores. No entanto, existem muitas tarefas para as quais é difícil ou mesmo impossível implementar, as quais podem ser divididas em quatro categorias gerais (Dietterich, 2003):

1. **Problemas para os quais não existem peritos humanos.** Por exemplo, nas fábricas modernas de produção autónomas, existe a necessidade de prever possíveis falhas antes que elas aconteçam, através da análise das leituras dos sensores. Devido ao facto das máquinas serem novas, não existe nenhum perito que possa ser entrevistado por um programador para fornecer conhecimento necessário para construir um sistema computacional. Um sistema de Aprendizagem Automática pode estudar dados gravados e falhas ocorridas para aprender as regras de previsão;
2. **Problemas onde existem peritos humanos,** mas onde estes são incapazes de explicar o seu conhecimento. Isto acontece em muitas tarefas de percepção, tal como reconhecimento de voz, reconhecimento de manuscritos e compreensão de linguagem natural. Felizmente, os peritos humanos podem fornecer exemplos de entrada (*inputs*) e de saída (*outputs*) correctos para estas tarefas, e assim sendo os algoritmos de aprendizagem automática podem aprender a mapear essas entradas com as saídas;
3. **Problemas em que os fenómenos mudam muito rapidamente.** Nas finanças, por exemplo, as pessoas gostariam de prever o comportamento futuro do mercado de acções, de compras dos consumidores, ou de taxas de câmbio. Estes comportamentos mudam frequentemente, mesmo que um programador possa construir um bom programa computacional de previsão, era necessário que fosse reescrito frequentemente. Um mecanismo de aprendizagem pode aliviar o programador dessa árdua tarefa por modificar constantemente o conjunto de regras de previsão aprendidas;
4. **Aplicações que precisam de ser personalizadas para cada utilizador separadamente.** Considere-se, por exemplo, um programa para filtrar mensagens electrónicas indesejáveis. Utilizadores diferentes requerem filtros diferentes. Um sistema de Aprendizagem Automática pode aprender quais as mensagens de correio electrónico que o utilizador rejeita e ajusta as regras de filtragem automaticamente.

A Aprendizagem Automática preocupa-se com as mesmas questões da Estatística, *Data Mining*, e Psicologia, mas com diferente ênfase. A Estatística foca-se em compreender os fenómenos que geram os dados, muitas vezes com o objectivo de testar diferentes hipóteses sobre esses fenómenos. Em *Data Mining* procura-se encontrar padrões nos dados que são compreendidos pelas pessoas. 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.). Pelo contrário, a Aprendizagem Automática preocupa-se principalmente com o desempenho do sistema computacional resultante. Para ilustrar isto, considere-se as diferentes questões que podem ser colocadas sobre dados de um discurso. Uma abordagem com Aprendizagem Automática foca-se em construir um sistema de reconhecimento de voz eficaz e eficiente. Uma abordagem estatística pode colaborar com uma abordagem psicológica para testar hipóteses sobre os mecanismos subjacentes ao reconhecimento de voz. Uma abordagem baseada em *Data Mining* pode procurar por padrões nos dados de forma a agrupar as pessoas de acordo com idade, sexo, ou nível de educação.

As áreas de aplicação de Aprendizagem Automática são várias, onde se inclui o processamento de linguagem natural, reconhecimento de padrões, análise de mercados, classificação de sequências de ADN, reconhecimento de voz e escrita, reconhecimento de objectos em visão por computador, jogos virtuais e locomoção robótica.

A literatura refere que, genericamente, os algoritmos e técnicas da Aprendizagem Automática estão divididos em três categorias: **Aprendizagem Supervisionada**, **Aprendizagem Não-Supervisionada** e **Aprendizagem por Reforço**. Estas categorias distinguem-se pelo tipo de informação: na Aprendizagem Supervisionada, os dados têm rótulos, ou classes; na Aprendizagem Não-Supervisionada, os dados não são rotulados; a Aprendizagem por Reforço distingue-se por ter o objectivo de maximizar uma recompensa (reforço). Por vezes é referida 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. É frequente também 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 à parte na secção 4.3 uma vez que se revelou bastante importante no desenvolvimento deste trabalho de mestrado.

4.2.1. Aprendizagem Supervisionada

A Aprendizagem Supervisionada é uma área da Aprendizagem Automática que se preocupa em criar uma função, ou regra de decisão, para efectuar uma previsão/decisão sobre novos dados, a partir de um conjunto de dados de treino. Estes dados de treino consistem em pares de objectos de entrada e respectivos dados de saída, tendo cada objecto a sua classe (rótulo) própria. A função criada pode gerar um valor contínuo (regressão) ou pode prever a classe do novo objecto (classificação). A tarefa do algoritmo supervisionado é prever o valor da função para qualquer objecto de entrada válido depois de analisar um conjunto de exemplos de treino.

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

O principal desafio da Aprendizagem Supervisionada é o problema da generalização: depois de analisar apenas um (normalmente pequeno) número de exemplos, o sistema de aprendizagem deve fornecer um classificador que funciona bem com qualquer exemplo possível.

O termo "Aprendizagem Supervisionada" inclui não só classificadores de aprendizagem, mas também funções de aprendizagem que prevêem valores numéricos (Dietterich, 2003).

4.2.2. Aprendizagem Não-Supervisionada

A Aprendizagem Não-Supervisionada é a área da Aprendizagem Automática cujo objectivo é 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, e o que não, geralmente acontece (Alpaydin, 2004). Em estatística, a Aprendizagem Não-Supervisionada é chamada de Estimativa de Densidade (*Density Estimation*). Um método de Aprendizagem Não-Supervisionada é o Agrupamento de Dados (C*lustering*) onde o objectivo é encontrar grupos nos dados de entrada (Alpaydin, 2004). Com o agrupamento dos dados, os objectos de um grupo são mais similares entre si comparativamente a elementos de grupos diferentes.

O termo "Aprendizagem Não-Supervisionada" é empregado para descrever uma vasta gama de diferentes tarefas de aprendizagem. Tal como o nome implica, estas tarefas analisam um dado conjunto de objectos que não têm uma classe associada (Dietterich, 2003).

4.2.3. Aprendizagem Semi-Supervisionada

Como já foi referido anteriormente, entre a Aprendizagem Supervisionada e a Aprendizagem Não-Supervisionada, existe a Aprendizagem Semi-Supervisionada, que se refere à classe de técnicas de Aprendizagem Automática que usa tanto dados rotulados como dados não-rotulados. Tipicamente é usada 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. A aquisição dos dados rotulados necessita de um agente humano para classificar os exemplos de treino manualmente. O custo associado com o processo de classificação de um conjunto de treino totalmente classificado pode tornar esse processo impraticável, considerando que a aquisição de dados não-rotulados é relativamente pouco dispendiosa. Nessas situações, a Aprendizagem Semi-Supervisionada pode ter um grande valor prático.

4.2.4. Aprendizagem por Reforço

A Aprendizagem por Reforço (*Reinforcement Learning*) é a área da Aprendizagem Automática que se preocupa em como um agente deve tomar acções num ambiente para maximizar uma recompensa (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 "aprendizagem de uma política óptima de mapeamento de situações para acções 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 correctos, nem acções sub-óptimas explicitamente correctas. Além disso, existe um foco no desempenho online, o qual envolve 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 acções serem executadas no ambiente (Panait & Luke, 2005).

A Aprendizagem-Q (*Q-Learning)* (Watkins & Dayan, 1992) e a Aprendizagem por Diferença Temporal *(Temporal Difference Learning)* (Sutton & Barto, 1998) são dois métodos referidos na literatura de Aprendizagem por Reforço. O primeiro aprende a utilidade de executar acções nos estados enquanto o segundo aprende, normalmente, a utilidade de estar nos estados. Os métodos de Aprendizagem por Reforço são inspirados por conceitos de programação dinâmica e definem fórmulas para actualizar as utilidades esperadas e para as usar na exploração do espaço de estados. A actualização é, muitas vezes, uma soma pesada do valor actual, o reforço obtido quando se executa uma acção, e a utilidade esperada do próximo estado alcançado após a acção ser executada. Enquanto se explora o espaço de estados, as estratégias determinísticas devem escolher as acções mais promissoras para serem executadas em cada estado (de acordo com as utilidades esperadas). Outras técnicas estocásticas podem levar a uma melhor exploração do espaço de estados. Os métodos de Aprendizagem por Reforço apresentam provas teóricas de convergência mas, infelizmente, muitas dessas suposições de convergência não se aplicam a muitas aplicações de mundo real (Panait & Luke, 2005).

Para mais informação acerca de técnicas de Aprendizagem por Reforço ver por exemplo (Barto et al., 1990)(Kaelbling et al., 1996)(Sutton, 1988).

4.3. Raciocínio baseado em Casos

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

Automática enquadra-se na Aprendizagem baseada em Instâncias, como já referido anteriormente. Opera sob a premissa que os problemas similares têm 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 as novas soluções".

Os problemas resolvidos anteriormente no sistema e as suas soluções ou estratégias associadas são memorizados como **casos**, guardados num repositório, a **base de casos**, para poderem ser reutilizados (Burke et al., 2002)(Beddoe et al., 2009). 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 retidos e a base de casos actualizada (Burke et al., 2002).

A descrição que se segue foi baseada na informação presente em (Kolodner, 1993) e (Aamodt & Plaza, 1994).

4.3.1. Motivação

O Raciocínio baseado em Casos é um paradigma de resolução de problemas que é bastante diferente em muitos aspectos 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. Uma segunda diferença importante é que o Raciocínio baseado em Casos é também 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 lembrança 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 diagnóstico para determinar a doença e aplica o mesmo tratamento no paciente à sua frente.

Tal como o exemplo anterior demonstra, o raciocínio através da 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[4] 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 duma forma que possa ser reutilizada na resolução de problemas futuros, é referida como um **caso passado**. Correspondentemente, 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 objectivos do sistema". Um caso consiste em duas partes principais (Burke et al., 2002): 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).

4.3.2. Perspectiva Histórica

As raízes do Raciocínio baseado em Casos em Inteligência Artificial são encontradas nos trabalhos de Roger Schank em memória dinâmica, assim como o papel central que a recordação de situações anteriores tem na resolução de problemas e aprendizagem (Schank, 1982). Também existem referências com base no estudo do raciocínio analógico (Gentner, 1983).

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

Outra base para o Raciocínio baseado em Casos, e outro conjunto de modelos, surgiu com Bruce Porter e o seu grupo (Porter & Bareiss, 1986). Inicialmente abordaram o problema de Aprendizagem Automática para tarefas de classificação. Desenvolveram então o sistema PROTOS

[4] Algo mais do que experimentar, remete para uma vivência.

que enfatizou a integração de conhecimento geral de domínio e conhecimento específico dos casos numa estrutura de representação unificada.

4.3.3. Aprendizagem no Raciocínio baseado em Casos

Uma característica muito importante no Raciocínio baseado em Casos é o seu acoplamento para aprendizagem. A força motriz por trás dos métodos baseados em casos surgiu da comunidade de Aprendizagem Automática, sendo o Raciocínio baseado em Casos um sub-campo da Aprendizagem Automática, também conhecido como Aprendizagem baseada em Casos. Deste modo, a noção de Raciocínio baseado em Casos não só denota um método de raciocínio particular, independentemente de como os casos são adquiridos, mas também denota um paradigma de Aprendizagem Automática que permite uma aprendizagem sustentada através da actualização da base de casos depois de um novo problema ter sido resolvido. Quando um problema é resolvido com sucesso, a experiência é retida de modo a ser possível resolver casos similares no futuro. Quando uma tentativa de resolução de um problema falha, a razão dessa falha é identificada e recordada para se evitar o mesmo erro no futuro.

O Raciocínio baseado em Casos favorece a aprendizagem a partir da experiência, uma vez que, normalmente, é mais fácil aprender através da retenção de uma experiência de resolução de um problema concreto do que generalizar a partir dele. Ainda, a aprendizagem efectiva no Raciocínio baseado em Casos requer um conjunto de métodos bem trabalhado para se extrair conhecimento relevante a partir da experiência, integrar o caso na estrutura de conhecimento existente, e indexar o caso para uso futuro em casos similares.

4.3.4. Fundamentos dos métodos de Raciocínio baseado em Casos

Todos os métodos de Raciocínio baseado em Casos lidam com tarefas centrais onde se 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 actualização do sistema através da aprendizagem da experiência.

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. Os casos podem ser mantidos como experiências concretas, ou um conjunto de casos similares devem dar origem a um caso generalizado. Os casos 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 directamente ao problema actual, ou modificada de acordo

com as diferenças entre os dois casos. A selecção de casos, adaptação de soluções, e aprendizagem a partir de uma experiência pode ser guiada e suportada por um modelo profundo de conhecimento geral de domínio, através de conhecimento mais compilado e raso, ou ser baseada apenas numa similaridade aparente e sintáctica. Os métodos de Raciocínio baseado em Casos podem ser puramente auto-suficientes e automáticos, ou podem interagir com o utilizador para apoio e orientação das suas acções. Alguns métodos de Raciocínio baseado em Casos 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.

4.3.5. O ciclo do Raciocínio baseado em Casos

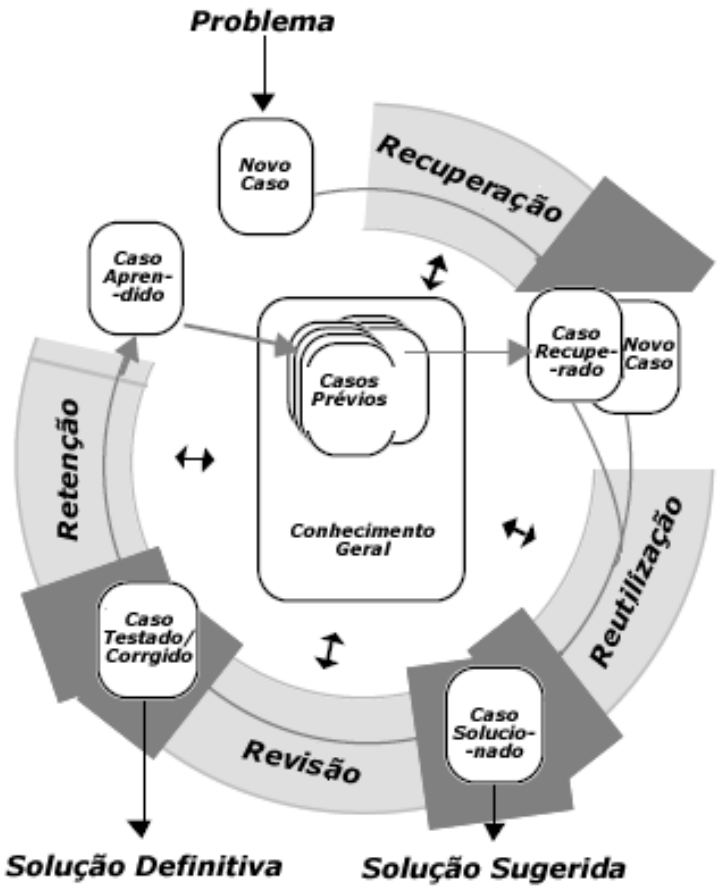

Figura 12 – O ciclo do Raciocínio baseado em Casos (adaptada de (Aamodt & Plaza, 1994))

Um ciclo geral de Raciocínio baseado em Casos pode ser descrito através dos quatro processos seguintes, conhecidos como “os quatro R’s” (Aamodt & Plaza, 1994)(Beddoe & Petrovic, 2006) (Beddoe et al., 2009):

1. **Recuperar** o caso, ou casos mais similares;
2. **Reutilizar** a informação e o conhecimento desse caso para resolver o problema;
3. **Rever** a solução proposta;
4. **Reter** as partes da experiência que se revelarem úteis para resoluções de problemas futuros.

Um novo problema é resolvido através da **recuperação** de um ou mais casos similares experimentados anteriormente, **reutilização** do caso mais similar, **revisão** da solução baseada na reutilização desse caso, e **retenção** da nova experiência através da incorporação do novo caso na base de conhecimento (base de casos). Este ciclo é ilustrado na Figura 12.

Uma descrição inicial de um problema (topo direito da Figura 12) define um **novo caso**. Este novo caso 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 um **caso solucionado**, ou seja, uma solução sugerida para o problema inicial. Através do processo de **revisão**, esta solução é testada, por exemplo, através da aplicação num ambiente de execução, e reparada se falhar. Durante o processo de **retenção**, a experiência útil é retida para reutilização futura, e a base de casos é actualizada com o novo **caso aprendido**, ou através da modificação de alguns casos existentes.

4.3.5.1. Representação de Casos

Um sistema de Raciocínio baseado em Casos é bastante dependente da estrutura e conteúdo da sua colecção de casos, ou 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 ser retida na base de casos, essa integração de um novo caso em memória deve ser igualmente eficaz e eficiente. O problema da representação no Raciocínio baseado em Casos é principalmente o problema de decidir como a base de casos deve ser organizada e indexada para uma recuperação e reutilização eficaz. Um problema adicional é como integrar a estrutura da base de casos num modelo de conhecimento geral de domínio, na medida em que tal conhecimento é incorporado.

4.3.5.2. Recuperação de Casos

A tarefa de recuperação começa com uma descrição parcial do problema, e acaba quando é encontrado o caso anterior mais similar. As suas subtarefas são referidas como a **Identificação**, **Correspondência**, e **Selecção**, executadas nessa ordem. A tarefa de Identificação surge basicamente com um conjunto de descritores relevantes para o problema, o objectivo da tarefa de Correspondência é retornar um conjunto de casos que são suficientemente similares com o novo

caso, dado uma similaridade mínima, e a tarefa de Selecção selecciona o melhor caso deste conjunto de casos.

Identificação

Identificar um problema pode envolver a simples observação dos seus descritores 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 dentro do seu contexto. Perceber um problema envolver a filtragem dos descritores ruidosos do problema, para inferir outras características relevantes do problema, para verificar se os valores da característica fazem sentido no contexto, para gerar expectativas para outras características, etc. Outros descritores podem ser inferidos 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.

Correspondência

A tarefa de encontrar uma boa correspondência é tipicamente dividida em duas subtarefas: um processo inicial de correspondência que recupera um conjunto de casos candidatos plausíveis, e um processo mais elaborado de selecção do melhor caso do conjunto.

Encontrar um conjunto de casos é feito através do uso dos descritores do problema (características de entrada) como índices para a base de casos de uma forma directa ou indirecta. Existem, em princípio, três formas de recuperação de casos: através do uso directo 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, e têm sido propostas várias métricas de similaridade, baseadas nas similaridades do problema com as características do caso.

Uma medida de similaridade é definida normalmente por uma fórmula para calcular a similaridade entre casos prévios e o novo caso (Burke et al., 2002). Os casos anteriores mais similares são recuperados para o novo caso. O desenvolvimento desta medida de similaridade para problemas complexos do mundo real é um dos maiores desafios de investigação nesta área.

Selecção

Nesta fase, é escolhido o melhor caso, a partir do conjunto de casos similares. Isto pode ser efectuado durante o processo de correspondência inicial, mas muitas vezes é retornado um conjunto

de casos, 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 selecção 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.

4.3.5.3. Reutilização de Casos

A reutilização da solução do caso recuperado foca-se em dois aspectos: as diferenças entre o caso passado e o caso actual, e na identificação de partes do caso recuperado que possam ser transferidas para o novo caso.

Cópia

Em tarefas de classificação simples as diferenças são abstraídas (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 directamente transferida para o novo caso, pois requer um processo de adaptação que tem em conta essas diferenças.

Adaptação

Existem duas formas de reutilizar casos passados: reutilizar a solução do caso passado e reutilizar o método passado que constrói a solução. No primeiro caso, a solução passada não é directamente a solução para o novo caso, mas existe algum conhecimento que permite transformar a antiga solução na solução para o novo caso. No segundo caso, 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-objectivos considerados, alternativas geradas, falhas, etc. O método recuperado é então reutilizado para o novo problema no novo contexto.

4.3.5.4. Revisão de Casos

Quando uma solução, para o caso, gerada pela fase de Reutilização não é correcta, 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 conseguida uma aprendizagem a partir do sucesso (passagem à próxima fase), senão é efectuada uma reparação da solução através da utilização de conhecimento específico do domínio do problema.

Avaliação da solução

A tarefa de 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 aparecer, dependendo do tipo de aplicação. A solução pode ainda ser aplicada num programa de simulação que é capaz de gerar a solução correcta.

Reparação de falhas

A reparação de casos envolve a detecção dos erros da solução actual 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 detectadas anteriormente não ocorram. A solução revista pode ser retida directamente, ou pode ser avaliada e reparada novamente.

4.3.5.5. Retenção de Casos

Este é o processo de 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. Envolve seleccionar qual a informação a reter do caso, em que forma retê-la, como indexar o caso para uma recuperação futura para casos similares, e como integrar o novo caso na estrutura de memória.

Extracção

No Raciocínio baseado em Casos, a base de casos é sempre actualizada, 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 nele. 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 dos casos, é necessário tomar uma decisão sobre o que usar como fonte de aprendizagem. Podem também ser incluídas no novo caso justificações acerca das soluções tomadas.

Indexação

O problema de indexação é um problema central, bastante focado no Raciocínio baseado em Casos, e trata de decidir quais os tipos de índices a usar em futuras recuperações de casos, bem como definir a estrutura do espaço de procura dos índices. Os índices directos 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 características de entrada como índices.

Integração

Corresponde ao último passo da actualização da base de conhecimento, 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 força, ou 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 para a 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.

4.3.6. Aplicações em Escalonamento

Nos últimos anos tem-se assistido a uma aceleração no interesse de desenvolvimento de sistemas de Raciocínio baseado em Casos numa variedade de problemas de optimização combinatória, onde se inclui o Escalonamento (Beddoe & Petrovic, 2006). Neste domínio do Escalonamento, alguns investigadores têm recorrido ao Raciocínio baseado em Casos para usar conhecimento/experiência memorizada na resolução de novos problemas. Burke et al. (2002) referiram que o Raciocínio baseado em Casos é uma abordagem bastante adequada para sistemas de Escalonamento periciais e enfatizaram um potencial de investigação em Escalonamento Dinâmico.

Na generalidade, as aplicações de Raciocínio baseado em Casos no domínio do 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 na primeira categoria é que é provável que uma abordagem que se revelou eficaz na resolução de um problema específico também será eficaz na resolução de um problema similar. Nestes sistemas, um caso consiste na representação do problema e num algoritmo conhecido que seja eficaz para a sua resolução. Schmidt (1998) desenhou uma estrutura de Raciocínio baseado em Casos para escolher o método mais apropriado para a resolução de um dado problema de Escalonamento 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 algoritmo apropriado. Schirmer (2000) implementou um sistema de Raciocínio baseado em Casos de selecção de algoritmos de Escalonamento para a resolução de problemas de escalonamento de projectos. Ele mostrou

experimentalmente que alguns algoritmos de Escalonamento funcionam melhor do que outros algoritmos em algumas instâncias de problemas. O sistema RBC pode seleccionar com sucesso um algoritmo e supera portanto um número de Meta-heurísticas. No entanto, as partes constituintes destas duas abordagens, ou seja a representação do caso (através de pares atributo-valor) e os casos recuperados são implementadas especialmente para o domínio de problema destes sistemas que são desenvolvidos. Portanto, estas abordagens não podem ser facilmente adaptadas e aplicadas aos problemas de calendarização.

Os sistemas de Raciocínio baseado em Casos para Escalonamento na segunda categoria reutilizam os operadores para a resolução do novo problema (Burke et al., 2002). 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). Burke et al. (2002) propuseram uma hiper-heurística baseada em casos para resolver problemas de calendarização. Um calendário é construído através da aplicação iterativa de um número de heurísticas, as quais podem ser seleccionadas por um controlador de Raciocínio baseado em Casos. Beddoe et al. (2009) desenvolveram um sistema de Raciocínio baseado em Casos para problemas de escalonamento de enfermeiras 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 da terceira categoria, 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. Este método tem sido aplicado para a resolução de problemas de escalonamento da produção (Coello & Santos, 1999) (MacCarthy & Jou, 1996) e problemas de calendarização de cursos universitários (Burke et al., 2002). Tem sido também usado para a construção de soluções iniciais para Meta-heurísticas, tais como os Algoritmos Genéticos (Oman & Cunningham, 2001) e *Simulated Annealing* (Cunningham & Smyth, 1997).

Existe também alguma investigação na área das hiper-heurísticas, embora alguns autores não usem esse termo. As hiper-heurísticas podem ser definidas como "heurísticas que escolhem heurísticas" ou como "algoritmos que escolhem o algoritmo adequado para determinada situação" (Burke et al., 2002). Uma abordagem foi apresentada por (Fang et al., 1994) em problemas de escalonamento *Open-Shop*[5] com o uso de Algoritmos Genéticos para procurar um espaço de abstracções de soluções para "evoluir a escolha da heurística". Os Algoritmos Genéticos têm sido usados para construir sistemas de escalonamento que escolhem as combinações óptimas das heurísticas (Hart et al., 1998). Burke et al. (2002) apresentaram um novo método de hiper-heurística usando Raciocínio baseado em Casos para resolver problemas de calendarização de cursos. A ideia básica é que mantêm uma base de casos de informação sobre as heurísticas mais bem sucedidas

[5] Problemas de escalonamento em que a ordem de processamento das operações nas diferentes máquinas é irrelevante.

para um grupo de problemas de calendarização conhecidos para prever a melhor heurística para um novo problema usando o conhecimento anterior. Técnicas de descoberta de conhecimento são usadas para realizar o treino do sistema de modo a melhorar o desempenho da previsão. Grolimund e Ganascia (1997) investigaram o uso de uma técnica de Inteligência Artificial para configurar uma Meta-heurística básica sem intervenção do utilizador. Assim, introduziram uma abordagem de Raciocínio baseado em Casos para realizar automaticamente o controlo do operador de selecção da Pesquisa Tabu.

4.4. Aprendizagem em Sistemas Multi-Agente

A inteligência implica um certo grau de autonomia o que requer a capacidade de tomar decisões de forma autónoma. Assim, os agentes devem possuir as ferramentas apropriadas para tomar tais decisões. Na maioria dos domínios dinâmicos não é possível prever todas as situações que um agente pode encontrar e portanto é necessário que o agente tenha a capacidade de se adaptar a novas situações. Isto verifica-se especialmente nos Sistemas Multi-Agente, onde em muitos casos o comportamento global emerge, em vez de ser pré-definido. Consequentemente, a aprendizagem é um componente crucial da autonomia e deve ser alvo de estudo dos agentes e dos Sistemas Multi-Agente (Alonso et al., 2001).

Nesta secção são abordados aspectos de aprendizagem para Sistemas de Fabrico baseados em agentes, discute-se a relação entre Aprendizagem Automática e Sistemas Multi-Agente, e referem-se alguns aspectos de aprendizagem de agentes bem como técnicas de aprendizagem.

4.4.1. Aprendizagem para Sistemas de Fabrico baseados em Agentes

A tecnologia baseada em agentes é considerada como uma abordagem importante para o desenvolvimento de sistemas avançados de fabrico, pois fornece estruturas dinâmicas reconfiguráveis e de resposta rápida para facilitar o uso flexível e eficiente dos recursos de fabrico num ambiente de rápida evolução (Shen et al., 1998).

Num Sistema Multi-Agente, é extremamente difícil ou até mesmo impossível determinar *a priori* o comportamento das suas actividades concretas correctamente. Isto requer, por exemplo, que se saiba *a priori* quais os requisitos de ambiente que vão surgir no futuro, quais os agentes que estarão disponíveis na altura, como é que esses agentes vão ter de interagir em resposta aos requisitos, etc. Tais problemas resultantes da complexidade dos Sistemas Multi-Agente podem ser evitados, ou pelo menos reduzidos, dotando os agentes da capacidade de aprender, isto é, com a capacidade de melhorar os desempenhos futuros do sistema global, ou de parte do sistema.

Assim, e tendo em vista a aprendizagem em sistemas de fabrico baseados em agentes, Shen, Maturana e Norrie (1998) identificaram quatro questões principais, descritas de seguida.

4.4.1.1. Porquê aprender?

Os ambientes de fabrico são sistemas dinâmicos de tempo-real. Existem sempre diferentes problemas tais como falhas nas máquinas, nas ferramentas ou nos veículos de transporte e falta de materiais necessários. Por outro lado, novas ferramentas, máquinas ou veículos de transporte podem ser adicionados ao ambiente de produção. Deste modo, um sistema de fabrico baseado em agentes deve ser capaz de se adaptar às mudanças no ambiente e lidar com contextos emergentes.

De acordo com Goldman e Rosenschein (1996), a aprendizagem em ambientes multi-agente pode ajudar os agentes a melhorar o seu desempenho. Os agentes, no relacionamento uns com os outros, podem aprender acerca do conhecimento e comportamentos estratégicos dos parceiros. Agentes que actuem em ambientes dinâmicos podem reagir a eventos inesperados por generalizar o que aprenderam na fase de treino. Nos sistemas cooperativos de resolução de problemas, o comportamento cooperativo pode ser feito 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 acções a partir do zero, para problemas semelhantes. Podem também ser capazes de, em algumas situações, evitar comunicações em tempo de execução dando uso a conceitos de coordenação já aprendidos, o que é especialmente útil quando os agentes não têm tempo suficiente para negociar.

4.4.1.2. Quando aprender?

Nos sistemas de fabrico baseados em agentes, os agentes devem aprender nas seguintes situações:

- **Quando a configuração do sistema muda:** por exemplo, novos recursos de fabrico (máquinas, ferramentas, veículos de transporte, trabalhadores, etc.) são adicionados ou removidos do sistema de trabalho; outro *hardware* ou *software* é alterado ou actualizado. Em qualquer dos casos mencionados anteriormente, cada agente deve aprender sobre as mudanças no sistema e actualizar o seu conhecimento sobre o ambiente e outros agentes;
- **Quando ocorrem algumas falhas no sistema:** podem ocorrer falhas tanto durante o processo de concepção como durante o teste/simulação do projecto. Ambas as situações representam oportunidades de aprendizagem;
- **Quando um projecto ou uma tarefa é terminada com sucesso:** em sistemas de fabrico, um plano bem sucedido (uma combinação eficiente dos recursos de fabrico para uma tarefa) é um bom caso de aprendizagem;

- **Quando é necessário melhorar as capacidades:** a necessidade de melhorar os projectos ou os planos de processo de fabrico pode ser traduzida num requisito para melhorar as capacidades dos agentes. Isto significa que um agente usa a aprendizagem sobre situações, disponibilidade de recursos de fabrico ou outros agentes para tomar decisões mais informadas no futuro. A observação de padrões, dependências ou outras relações pode ser útil embora não seja motivada pelos acontecimentos das situações anteriores.

4.4.1.3. Onde aprender?

Nos sistemas de fabrico baseados em agentes, a forma como um agente coloca a informação disponível para os outros agentes determina onde as oportunidades de aprendizagem surgem. Variando a representação e o contexto no qual o conhecimento é comunicado favorece alguns tipos de aprendizagem em comparação com outros.

A comunicação directa implica a transferência de conhecimento entre os agentes através de ligações directas. O conhecimento é encapsulado numa mensagem com um formato especial e é enviado para os outros agentes. O conhecimento transferido não é alterado neste processo. Quando um agente recebe este tipo de mensagem, extrai a informação que considera útil e actualiza as suas bases de conhecimento.

A comunicação indirecta assume a presença de um agente intermediário, que transmite a informação de um agente para outro.

Alguns Sistemas Multi-Agente de aprendizagem são implementados com agentes de treino (agentes tutores). Neste caso, os agentes de treino, ou tutores, são as principais fontes de conhecimento de aprendizagem do sistema.

4.4.1.4. O que aprender?

O conhecimento que pode e deve ser aprendido em sistemas de fabrico baseados em agentes inclui os êxitos e os fracassos do passado, a utilidade de diferentes peças de conhecimento com respeito a diferentes tarefas e situações, as capacidades dos outros agentes no sistema, e o relacionamento entre o Sistema Multi-Agente e o seu ambiente.

Enquanto o leque de aprendizagem é bastante vasto, existem áreas onde a aprendizagem pode ter um sério impacto na indústria e nas interacções entre os agentes. Alguns dos seguintes tipos de aprendizagem focam o nível de alocação de recursos de fabrico, enquanto outros abordam as interacções entre agentes:

- **Combinação de recursos de fabrico para tarefas específicas:** Estas combinações de recursos ao nível do grupo ou ao nível do sistema podem ser aprendidas como casos de

planeamento de processos de fabrico, os quais tanto podem ser utilizados nos planos e escalonamento de processos seguintes como para reduzir as comunicações e negociações entre os agentes recurso;

- **Comportamento do sistema de fabrico:** Através da aprendizagem do comportamento do sistema e propagando esse conhecimento para o futuro próximo através de alguns mecanismos de simulação de previsões, é permitido ao sistema aprender a partir de comportamentos emergentes futuros de modo a prevenir as perturbações e mudanças imprevistas nas prioridades de produção;
- **Apoio em favor de ou contra uma decisão:** O raciocínio/argumentação pode ajudar principalmente para analisar propostas de diferentes agentes e decidir qual o agente que deve rever a sua decisão;
- **Regras:** Como os agentes têm funcionalidades e pontos de vista diferentes, é desejável a partilha de regras entre agentes que tenham o mesmo objectivo ou pelo menos o mesmo domínio;
- **Preferências:** Se as preferências dos agentes são conhecidas, então ajudam a estabelecer propostas iniciais mais adequadas, feitas por um agente a outro;
- **Pré-condições e pós-condições para regras, acções e tarefas:** Este tipo de aprendizagem é importante para uma melhor adaptação de decisões num contexto específico. Os agentes decidem por eles próprios quando devem agir, e portanto devem ser capazes de reconhecer situações favoráveis para serem envolvidas no projecto;
- **Previsão das decisões dos outros agentes:** Isto equivale a construir um modelo comportamental dos outros agentes, ao relacionar factores que influenciam um agente com as decisões dos agentes;
- **Tipos de conflitos:** Os conflitos podem ser detectados, indexados e classificados de forma diferente pelos diversos tipos de agentes. Permitindo aos agentes classificar e reconhecer conflitos melhora a antecipação e mecanismos de resolução dos mesmos;
- **Heurísticas para resolver conflitos e para negociar:** Estas heurísticas, ao serem menos dependentes das funcionalidades específicas de um agente, podem ser aprendidas ou transferidas de um agente para outro e refinadas de acordo com as necessidades.

4.4.2. A relação entre Aprendizagem Automática e Sistemas Multi-Agente

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 & Luke, 2005). Embora tenha vindo a ser estudada extensivamente ao longo dos anos, tem sido na sua maioria investigada separadamente dos agentes

e só recentemente recebeu maior atenção por parte da área dos Sistemas Multi-Agente (Alonso et al., 2001).

Os Sistemas Multi-Agente apresentam o problema da aprendizagem distribuída, isto é, muitos agentes a aprenderem separadamente para atingir um resultado global comum. Os algoritmos de aprendizagem existentes 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, tais algoritmos de aprendizagem precisam de modificações, ou então é necessário que sejam desenvolvidos novos algoritmos. Na aprendizagem distribuída, os agentes precisam de cooperar e comunicar para aprenderem efectivamente (Alonso et al., 2001).

Panait e Luke (2005) focaram-se na aplicação da Aprendizagem Automática para problemas da área de Sistemas Multi-Agente. 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. A Aprendizagem Automática provou ser uma abordagem popular para a resolução de problemas de Sistemas Multi-Agente porque a complexidade inerente de muitos desses problemas pode fazer com que as soluções sejam proibitivamente difíceis. Focaram especificamente os domínios de problemas onde os múltiplos agentes cooperam para resolver uma tarefa, designada **Aprendizagem Cooperativa Multi-Agente**.

Como referido anteriormente, existem três abordagens principais de aprendizagem: Supervisionada, Não-Supervisionada e baseada em Recompensa. Estes métodos distinguem-se no tipo de retorno que é fornecido ao aprendiz: na Aprendizagem Supervisionada é fornecida a informação correcta, na Aprendizagem Não-Supervisionada não é dado qualquer *feedback*, e na Aprendizagem baseada em Recompensa é fornecida uma avaliação da qualidade (a “recompensa”) da informação do aprendiz.

Devido à complexidade inerente nas interacções entre os múltiplos agentes, vários métodos de Aprendizagem Automática (nomeadamente os métodos de Aprendizagem Supervisionada) não são facilmente aplicáveis ao problema porque assumem tipicamente a existência de um “crítico” que pode fornecer aos agentes o comportamento “correcto” para uma dada situação. Algumas excepções são encontradas em (Garland & Alterman, 2004) (Goldman & Rosenschein, 1996)(Williams, 2004). Assim, têm sido maioritariamente usados métodos de Aprendizagem baseada em Recompensa.

A literatura em Aprendizagem baseada em Recompensa encontra-se maioritariamente dividida em dois subconjuntos: métodos de Aprendizagem por Reforço, que estimam funções de valor, e métodos de Pesquisa Estocástica (Spall, 2003), tais como Computação Evolucionária, *Simulated Annealing*, entre outros, que aprendem comportamentos sem requerer funções de valor. Na literatura da área de Pesquisa Estocástica, a maioria da discussão multi-agente concentra-se na Computação Evolucionária (Panait & Luke, 2005):

- **Aprendizagem por Reforço:** Os métodos de Aprendizagem por Reforço apresentam provas teóricas de convergência mas, infelizmente, muitas dessas suposições de convergência não se aplicam a muitas aplicações de mundo real, incluindo muitos problemas de Sistemas Multi-Agente. Para mais informação acerca de técnicas de Aprendizagem por Reforço ver (Barto et al., 1990)(Kaelbling et al., 1996)(Sutton, 1988);
- **Computação Evolucionária:** A Computação Evolucionária (ou Algoritmos Evolucionários) é uma família de técnicas nas quais modelos *Darwinianos* abstractos de evolução são aplicados para refinar populações de soluções candidatas (conhecidos como "indivíduos") a um dado problema. Um algoritmo evolucionário começa com uma população inicial de indivíduos gerados aleatoriamente. Cada membro desta população é então avaliado e é-lhe atribuído um valor de aptidão (*fitness*) que representa uma avaliação de qualidade. O Algoritmo Evolucionário usa então um procedimento orientado ao valor de aptidão para seleccionar, criar e modificar indivíduos para produzir filhos que são adicionados à população, substituindo indivíduos antigos. Um ciclo de avaliação, selecção e criação é conhecido como "geração". Sucessivas gerações continuam a refinar a população até atingir o critério de paragem temporal ou até um indivíduo suficientemente bom ser encontrado. Os métodos de computação evolucionária incluem os Algoritmos Genéticos e as Estratégias Evolutivas, sendo ambos aplicados à pesquisa de espaços de parâmetros multidimensionais, e a Programação Genética que se preocupa com a evolução dos programas actuais. Existem muitos recursos com informação adicional sobre Computação Evolucionária, entre os quais (Bäck, 1996)(Jong, 2005)(Michalewicz, 1996).

Os Algoritmos Co-Evolucionários representam uma abordagem natural na aplicação da Computação Evolucionária para refinar comportamentos multi-agente. Nestes algoritmos, o *fitness* de um indivíduo é baseado nas interacções com os outros indivíduos da população: assim a avaliação do *fitness* é sensível ao contexto e subjectiva. Na Co-Evolução Competitiva os indivíduos beneficiam à custa uns dos outros, mas na Co-Evolução Cooperativa os indivíduos têm sucesso ou falham todos juntos na colaboração. Uma abordagem normal de aplicação de Algoritmos Co-Evolucionários Cooperativos a um problema de optimização começa por decompor a representação do problema em sub-componentes, seguindo-se a atribuição de cada sub-componente a uma população separada de indivíduos (Potter, 1997)(Potter & Jong, 2000).

4.4.3. Aspectos de Aprendizagem de Agentes

Seguidamente serão descritos alguns aspectos da aprendizagem dos agentes, nomeadamente a distinção entre Aprendizagem de Agente Único e Aprendizagem Multi-Agente, assim como a referência a métodos de Aprendizagem On-line e Off-line.

4.4.3.1. Aprendizagem de Agente Único vs. Aprendizagem Multi-Agente

De acordo com Stone e Veloso (1997), a Aprendizagem de Agente Único foca-se em como um agente melhora as suas capacidades individuais, independentemente do domínio onde está inserido. Normalmente consiste em métodos de Aprendizagem Automática e é a base para a Aprendizagem Multi-Agente. Por um lado, a Aprendizagem de Agente Único pode nem sempre levar a um desempenho óptimo em ambientes multi-agente e podem existir domínios em que a Aprendizagem Multi-Agente Coordenada é mais natural e mais eficaz. No entanto, não é possível falar-se acerca de Aprendizagem Multi-Agente se o que um agente aprende não afecta ou não é afectado pelos outros agentes vizinhos (Alonso et al., 2001).

Define-se genericamente a Aprendizagem Multi-Agente como a aplicação da Aprendizagem Automática a problemas que envolvem múltiplos agentes. Panait e Luke (2005) referem duas características da Aprendizagem Multi-Agente que requerem o seu estudo fora da Aprendizagem Automática. Primeiro, porque a Aprendizagem Multi-Agente lida com domínios de problemas que envolvem múltiplos agentes, o espaço de procura pode ser invulgarmente grande, e, devido à interacção entre esses agentes, pequenas mudanças nos comportamentos aprendidos podem resultar muitas vezes em alterações imprevisíveis nas propriedades do Sistema Multi-Agente como um todo. Segundo, a Aprendizagem Multi-Agente pode envolver muitos aprendizes, cada um a aprender e a adaptar-se no contexto dos outros. Isto introduz aspectos de teoria de jogos ao processo de aprendizagem os quais ainda não estão totalmente consolidados.

Os algoritmos de Aprendizagem Multi-Agente são então, na sua maioria, baseados nos algoritmos de Aprendizagem de Agente Único. A distinção reside na existência de outros agentes e interacções na Aprendizagem Multi-Agente, pois um agente pode aprender conhecimento a partir de outros agentes bem como a partir do ambiente. Os objectivos da Aprendizagem Multi-Agente podem ser a coordenação, a aprendizagem dos modelos e intenções dos outros agentes, a comunicação efectiva com outros agentes, entre outros, sendo que nenhum destes objectivos existe em Aprendizagem de Agente Único. No entanto, do ponto de vista algorítmico, a Aprendizagem Multi-Agente não parece ser muito diferente da Aprendizagem de Agente Único. Seguidamente são apresentados alguns desses algoritmos[6]:

- **Aprendizagem Multi-Agente por Reforço:** A Aprendizagem Multi-Agente por Reforço é implementada de várias formas. O termo é amplamente usado na literatura mas o seu significado varia entre os investigadores. Uma definição estende simplesmente o par estado/acção de agentes únicos para agentes múltiplos, ou partilha uma politica comum entre os vários agentes. Por exemplo, a definição de Hu e Wellman (1998) combina duas acções de agentes como uma acção única da tradicional Aprendizagem por Reforço. Pendrith (2000) apresenta um algoritmo de Aprendizagem por Reforço distribuída onde o retorno dos vários agentes é actualizar simultaneamente uma política única partilhada.

[6] Descritos em http://www.geocities.com/zijian_ren/research/SurveyMAL.pdf

Uma segunda definição de Aprendizagem Multi-Agente por Reforço estende o âmbito do ambiente, no qual um agente considera os comportamentos dos outros agentes como uma extensão do seu ambiente. Por exemplo, Mataric (1995) usa um método de Aprendizagem por Reforço modificado, cuja recompensa é a combinação do ambiente e dos outros agentes. Uma terceira definição é relacionada com as interacções dos agentes, na qual os agentes aprendem através da troca de informação tal como sensações, políticas ou valores. Tan (1993) estudou os agentes cooperativos de três formas: partilha de sensações, partilha de episódios e partilha de políticas aprendidas comparando com agentes independentes. As suas experiências mostraram que a partilha de políticas ou episódios aumenta a aprendizagem com o custo de mais comunicação;

- **Aprendizagem Multi-Agente baseada em Regras:** A Aprendizagem baseada em Regras pode ser facilmente transferida para Sistemas Multi-Agente através da integração das regras dos outros agentes com as suas próprias regras. Apesar de ser de implementação simples, a Aprendizagem Multi-Agente baseada em Regras não é um método de aprendizagem muito popular. Pode não funcionar bem num ambiente multi-agente, o qual é incerto e flexível, mas, no entanto, isto não significa que a Aprendizagem baseada em Regras não seja apropriada para Aprendizagem Multi-Agente. A solução pode residir no desenvolvimento de novos sistemas de regras integrando outros métodos de aprendizagem. O sistema de raciocínio humano possibilita a integração de regras com Redes Neuronais;
- **Outros métodos de Aprendizagem Multi-Agente:** Podem ser considerados nesta categoria os métodos baseados em Redes Neuronais e também os algoritmos evolucionários. Quando os agentes trocam alguns elementos cruciais dos seus métodos estatísticos, a aprendizagem estatística pode ser estendida para Aprendizagem Multi-Agente. Também a Aprendizagem baseada em Casos pode ser facilmente implementada nos Sistemas Multi-Agente.

A hierarquia pode ser o factor mais importante durante o desenvolvimento de Aprendizagem Multi-Agente. A hierarquia de aprendizagem pode ser considerada a dois níveis: hierarquia de algoritmos de aprendizagem e hierarquia organizacional de agentes. A Aprendizagem Multi-Agente com apenas um método simples 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 acções dos agentes.

Além dos algoritmos descritos, também é possível ver a Aprendizagem Multi-Agente numa perspectiva cooperativa, recaindo na **Aprendizagem Cooperativa Multi-Agente** (Panait & Luke, 2005)**.** Enquanto alguns podem definir a cooperação como uma recompensa partilhada, muita da literatura tende a usar uma noção de cooperação muito mais abrangente, particularmente com a

introdução de atribuição de crédito (ver subsecção 4.4.4.4). Como tal, se o sistema de aprendizagem é construído de forma a encorajar a cooperação entre os agentes, então é suficiente, mesmo se no final os agentes falharem nessa cooperação. Panait e Luke (2005) acreditam que existem duas categorias principais de abordagens de Aprendizagem Cooperativa Multi-Agente. A primeira, chamada **Aprendizagem em Equipa** (subsecção 4.4.4.3), aplica um único aprendiz para procurar por comportamentos para a comunidade de agentes. A segunda categoria de técnicas, **Aprendizagem Concorrente** (subsecção 4.4.4.4), utiliza múltiplos processos de aprendizagem concorrente, e, embora a sua designação pareça antagónica relativamente à Aprendizagem Cooperativa, o seu objectivo passa igualmente pela cooperação e melhoria do desempenho global do Sistema Multi-Agente. 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 projectando 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.

Para ilustrar a diferença entre as duas abordagens, considere-se novamente o cenário de exploração cooperativa, onde o objectivo é maximizar o número de itens recolhidos do ambiente. Uma abordagem de Aprendizagem em Equipa emprega um único aprendiz para, iterativamente, melhorar o "comportamento de 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 – exactamente quantos itens cada explorador recolheu). Os agentes exploradores melhoram os seus comportamentos independentemente uns dos outros, mas não têm qualquer controlo de como os outros decidem comportar-se.

4.4.3.2. Métodos de Aprendizagem On-line e Off-line

Algoritmos de Aprendizagem On-line (ou incremental), tais como redes neuronais de propagação de erro ou, em alguns casos, Aprendizagem por Reforço, têm vindo a ser usados para computar de forma incremental novas hipóteses assim que novos exemplos de treino ficam disponíveis. Por outro lado, os métodos de Aprendizagem Off-line induzem hipóteses a partir de um conjunto de exemplos de treino apresentados ao algoritmo num único ponto no tempo. Obviamente, os métodos on-line são mais apropriados para os Sistemas Multi-Agente onde os agentes precisam de actualizar constantemente o seu conhecimento, mas pode ser também desejável usar os algoritmos off-line mais poderosos. Para isto, um agente precisa de recolher um conjunto de exemplos de teste e decidir quando computar uma hipótese. Os maiores problemas a serem resolvidos são decidir quais os

exemplos de treino a coleccionar (e qual o formato que devem ter) e quando se deve executar o algoritmo de aprendizagem (Alonso et al., 2001).

4.4.4. Técnicas de Aprendizagem

Nesta secção serão descritas várias técnicas de aprendizagem usadas em Sistemas Multi-Agente, nomeadamente a Aprendizagem em Sistemas Reactivos, a Aprendizagem Social, a Aprendizagem em Equipa, e a Aprendizagem Concorrente.

4.4.4.1. Aprendizagem em Sistemas Reactivos

Nos sistemas reactivos, o comportamento conjunto emerge da interacção dos comportamentos dos agentes. Em vez de se implementar protocolos de coordenação ou de se fornecer modelos de reconhecimento complexos, é assumido que os agentes funcionam com informação baseada em valor (tal como a distância que devem manter dos seus vizinhos) que produz o comportamento social. Uma vez que o processamento interno é evitado, estas técnicas permitem aos sistemas de agentes responder rapidamente às mudanças (Alonso et al., 2001).

Como um efeito colateral, os agentes não possuem conhecimento de domínio que é essencial para tomar decisões em cenários dinâmicos e complexos. Não é possível simular interacções sociais complexas (mercados, conflitos, etc.) assumindo que os agentes não possuem qualquer conhecimento de domínio do seu ambiente. Para possuírem comportamentos de alto-nível, os agentes precisam de resumir as suas experiências em conceitos. A entidade que pode conceptualizar pode criar experiência em conhecimento e orientar os recursos vitais até que sejam requeridos (Alonso et al., 2001).

Mesmo que sejam principalmente técnicas off-line, as técnicas de Aprendizagem baseadas em Conhecimento tais como Aprendizagem baseada em Explicações e a Programação de Lógica Indutiva são ferramentas adequadas para resolver as limitações dos sistemas reactivos.

Na **Aprendizagem baseada em Explicações** (Carbonell et al., 1991) os agentes preocupam-se mais com a melhoria da eficácia do que como adquirir novo conhecimento. Obviamente, os agentes, quando apresentados com o mesmo problema repetidamente, não o devem resolver da mesma forma e no mesmo tempo de execução. Pelo contrário, parece sensível usar conhecimento genérico para analisar, ou explicar, cada instância do problema para optimizar o desempenho futuro. Esta aprendizagem não é meramente uma forma de fazer o programa mais rápido, mas uma forma de produzir mudanças qualitativas no desempenho de um sistema de resolução de problemas. A Aprendizagem baseada em Explicações extrai regras genéricas a partir de exemplos únicos gerando uma explicação porque é que o sistema foi bem ou mal sucedido, e generalizando-a. Isto fornece um

método dedutivo para tornar conhecimento em perícia útil e eficiente. As regras aprendidas permitem que seja feita a escolha acertada se uma situação similar surge durante a resolução de problemas.

A **Programação de Lógica Indutiva** (Muggleton & Raedt, 1994) computa uma hipótese baseada em circunstâncias externas e inicialmente desconhecidas. Em domínios de mundo-real, utilizar somente Aprendizagem baseada em Explicações pode ser impraticável pois os agentes trabalham com conhecimento incompleto, e portanto a Programação de Lógica Indutiva pode ser importante para a eficácia do sistema. Os métodos de Programação de Lógica Indutiva computam uma hipótese indutiva com a ajuda de dados treinados (exemplos positivos e negativos) e conhecimento de *background*. Os agentes coleccionam os exemplos de treino baseados nas acções executadas e nas consequências. Isto, juntamente com a base de conhecimento de domínio e um predicado alvo (por exemplo, sucesso ou falha) forma o básico do processo de aprendizagem indutiva. Quando um certo número de exemplos de treino é classificado incorrectamente, uma nova hipótese será computada baseada no conjunto de treino prolongado.

4.4.4.2. Aprendizagem Social

A Aprendizagem Social é composta por mecanismos de aprendizagem com origem na Inteligência Artificial e na Biologia que podem ser vistos como uma alternativa às abordagens baseadas em lógica apresentadas anteriormente.

Considere-se um Sistema Multi-Agente persistente, onde novos agentes entram num mundo já populado com agentes experientes. Um novo agente começa com um estado em branco, e ainda não teve oportunidade de aprender acerca do seu ambiente. No entanto, um novo agente pode não precisar de descobrir tudo acerca do seu ambiente: pode beneficiar da aprendizagem acumulada da população de agentes mais experientes.

Uma diferença importante entre os agentes artificiais e os animais é que no primeiro caso é possível forçar frequentemente um cenário completamente cooperativo, onde o que é bom para um agente é bom para todos (isto é, uma função de utilidade comum). Embora a cooperação ocorra em muitas espécies animais, o potencial para conflitos nunca está ausente, por causa da competição entre a auto-replicação dos genes no centro do processo evolucionário. Então, a Aprendizagem Social Cooperativa pode ser mais fácil de manter, e simples de perceber, numa população de agentes de *software* do que em espécies reais. No entanto, os conflitos de interesse continuam a ser relevantes se os agentes estão a operar em ambientes com competidores potencialmente maliciosos (por exemplo, na Internet). A Aprendizagem Social num caso destes pode envolver a complicação adicional de se ter a certeza de que o “professor” não está a tentar ludibriar o aprendiz a fim de satisfazer os seus próprios interesses (Alonso et al., 2001).

Existem muitas formas de um agente poder aprender a partir dos comportamentos dos outros. Apesar de existir desde há muito um foco na imitação (isto é, uma cópia directa dos comportamentos dos outros), esta revelou-se complexa uma vez que envolve não apenas perceber e reproduzir os

movimentos corporais dos outros, mas perceber também as mudanças no ambiente provocadas pelos comportamentos dos outros, e ser capaz de captar as "relações intencionais" entre estes, isto é, saber como e porquê o comportamento deve levar a um determinado objectivo (Alonso et al., 2001). Assim, seguidamente são apresentados alguns mecanismos mais simples que podem ser facilmente implementados em agentes robóticos ou de *software*, descritos por Alonso et al. (2001):

- **Comportamentos contagiosos:** são exemplificados por uma regra tal como "se fogem, eu também fujo". A ideia é que os estímulos produzidos pelo desempenho de um comportamento particular funcionam como gatilhos para os outros se comportarem da mesma forma. Note-se que isto não envolve aprendizagem real, e é meramente um sistema reactivo, mas pode todavia produzir comportamentos sociais adaptativos. Rir e bocejar são bons exemplos de comportamentos contagiosos nos humanos;
- **Aprimoramento de estímulos:** também chamado de aprimoramento local, é o que acontece quando os animais obedecem a uma regra tal como "seguir alguém mais velho do que nós, e aprender com o que acontecer". Uma tendência comportamental simples – neste caso, seguir alguém da mesma espécie – combina com a capacidade de aprender para resultar numa potencial transmissão dos comportamentos adquiridos;
- **Aprendizagem empírica:** quando se adiciona capacidades de aprendizagem mais sofisticadas ao aprimoramento de estímulos. O algoritmo envolvido é aproximadamente "prestar atenção ao que os outros estão a fazer, e se os resultados parecerem bons ou maus para eles, aprender com isso".

4.4.4.3. Aprendizagem em Equipa

Na Aprendizagem em Equipa, existe um único aprendiz envolvido, mas com o objectivo de descobrir um conjunto de comportamentos para uma equipa de agentes, em vez de um único agente. É uma simples abordagem à Aprendizagem Multi-Agente porque o aprendiz pode usar técnicas padrão de Aprendizagem Automática de agente único. Isto contorna as dificuldades emergentes da co-adaptação de múltiplos aprendizes que se encontram em abordagens de Aprendizagem Concorrente. Outra vantagem de um aprendiz único é que este se preocupa com o desempenho da equipa, e não só consigo próprio. Por esta razão, as abordagens de Aprendizagem em Equipa podem ignorar a atribuição de crédito inter-agente o que é normalmente difícil de determinar (Panait & Luke, 2005).

Mas a Aprendizagem em Equipa apresenta algumas desvantagens (Panait & 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 de 100 estados e o agente *B* pode estar noutros 100 estados, a equipa formada pelos dois agentes pode estar em qualquer de 10.000 estados (100x100). Esta explosão do tamanho do espaço de estados pode ser esmagadora para métodos de aprendizagem que exploram o espaço de utilidades de estados (tal como Aprendizagem por Reforço) mas pode não afectar tão drasticamente as técnicas que exploram o espaço de comportamentos (tal

como a Computação Evolucionária). Uma segunda desvantagem refere-se à centralização do algoritmo de aprendizagem: todos os recursos necessitam de estar disponíveis num único sítio onde o programa irá ser executado. Isto pode ser incómodo em domínios onde os dados estão inerentemente distribuídos.

A Aprendizagem em Equipa pode ainda ser **homogénea** ou **heterogénea** (Panait & Luke, 2005). Os aprendizes homogéneos desenvolvem um comportamento único por cada agente. Os aprendizes heterogéneos devem lidar com um elevado espaço de procura, mas com a promessa de obter melhores soluções através da especialização de agentes. Existem abordagens entre as duas categorias que são chamadas de métodos híbridos de Aprendizagem em Equipa. A escolha entre as abordagens depende se são ou não necessários especialistas na equipa:

- **Aprendizagem Homogénea:** todos os agentes apresentam comportamentos idênticos, mesmo que os agentes não sejam idênticos (por exemplo, agentes diferentes podem levar tempo diferente para completar a mesma tarefa). Devido a todos os agentes terem o mesmo comportamento, o espaço de procura para o processo de aprendizagem é drasticamente reduzido. A conveniência da aprendizagem homogénea depende do problema: alguns problemas não requerem a especialização de agentes para atingir um bom desempenho. Para outros domínios do problema, particularmente aqueles com um grande número de agentes, o espaço de procura é demasiado grande para se usar aprendizagem heterogénea, mesmo se a heterogeneidade produzisse os melhores resultados. Os agentes podem agir heterogeneamente mesmo em Aprendizagem Homogénea em Equipa, se os comportamentos homogéneos especificarem sub-comportamentos que diferem baseados numa condição inicial dos agentes (localização, etc.) ou no seu relacionamento com outros agentes;
- **Aprendizagem Heterogénea:** a equipa é composta por agentes com comportamentos diferentes, com um único aprendiz a tentar melhorar a equipa como um todo. Esta abordagem permite maior diversidade da equipa no aumento do espaço de procura. A maior parte da investigação em Aprendizagem Heterogénea em Equipa preocupa-se com o requisito para o aparecimento de especialistas. A tão chamada co-evolução de “uma população” pode ser usada com Aprendizagem Heterogénea em Equipa de uma forma pouco comum: uma população única é desenvolvida usando um algoritmo evolucionário ordinário, mas os agentes são testados emparelhando-os com outros escolhidos aleatoriamente da mesma população, para formar uma equipa heterogénea (Bull & Holland, 1997) (Quinn, 2001);
- **Aprendizagem Híbrida:** o conjunto de agentes é dividido em vários pelotões, pertencendo cada agente a um único pelotão. Todos os agentes de um pelotão têm o mesmo comportamento. Um extremo (um pelotão único) é equivalente a Aprendizagem Homogénea em Equipa, enquanto o outro extremo (um agente por pelotão) é equivalente a Aprendizagem Heterogénea em Equipa. A Aprendizagem Híbrida em Equipa permite dessa forma conhecer algumas das vantagens de cada método (Panait & Luke, 2005).

4.4.4.4. Aprendizagem Concorrente

A alternativa mais comum à Aprendizagem em Equipa em Sistemas Multi-Agente 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 ser dividida em "pelotões", cada um com o seu próprio aprendiz (Panait & Luke, 2005).

A principal questão na Aprendizagem Concorrente é saber quando é que esta será mais adequada que a Aprendizagem Homogénea ou Heterogénea em Equipa. Jansen e Wiegand (2003) argumentam que a Aprendizagem Concorrente pode ser preferível nos domínios em que 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 projecta 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 são 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 partir o processo de aprendizagem em pedaços mais pequenos permite uma maior flexibilidade no uso dos recursos computacionais na 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 na volta pode arruinar os comportamentos aprendidos pelos outros agentes fazendo ficar as suposições obsoletas naquilo que são baseadas (Sandholm & Crites, 1995)(Weinberg & Rosenschein, 2004). Uma abordagem simples para lidar com esta co-adaptação é tratar os outros aprendizes como parte do ambiente dinâmico para o qual dado aprendiz se deve adaptar (Schmidhuber, 1996) (Schmidhuber & Zhao, 1996). Mas, em Aprendizagem Concorrente, as coisas tornam-se mais complicadas pois os outros agentes não estão meramente em mudança, mas sim a co-adaptar-se à adaptação prévia do aprendiz a eles próprios. Então, a adaptação dos agentes ao ambiente pode muda-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.

Panait e Luke (2005) argumentam que existem três temas principais na área da Aprendizagem Concorrente. Primeiro, existe o problema da **atribuição de crédito**, que trata da partilha da recompensa obtida pela equipa para os aprendizes individuais. Segundo, existem os desafios nas **dinâmicas de aprendizagem** que permitem aprender o impacto da co-adaptação nos processos de

aprendizagem. Por fim, a **modelização dos outros agentes** a fim de melhorar as interacções (e colaborações) entre eles.

Atribuição de Crédito

Quando se lida com múltiplos aprendizes, surge a necessidade de dividir entre eles a recompensa recebida através das suas acções conjuntas. A solução mais simples é dividir essa recompensa de equipa de igual forma pelos agentes, ou num sentido mais amplo, dividir a recompensa para que, quando a recompensa de um aprendiz aumenta (ou diminui), todas as recompensas de todos os agentes aumentem (ou diminuem). Esta abordagem de atribuição de crédito é conhecida como **recompensa global**. No entanto, existem várias situações onde pode ser desejável atribuir crédito de maneira diferente. Se certos agentes cumpriram com as suas obrigações, talvez seja proveitoso recompensar especialmente esses agentes ou castigar os outros por preguiça. Se não se divide de igual forma a recompensa pelos agentes, existe, num extremo, a necessidade de estimar o desempenho de cada agente baseado somente no seu comportamento individual. Esta abordagem desencoraja a preguiça porque recompensa os agentes apenas pelas tarefas que eles realmente cumpriram. No entanto, os agentes não têm qualquer incentivo racional para ajudar os outros agentes, e podem ser desenvolvidos comportamentos gulosos. Esta abordagem é chamada de **recompensa local**;

Dinâmicas da Aprendizagem

Quando se aplica Aprendizagem de Agente Único a ambientes estacionários, o agente experimenta comportamentos diferentes até, esperançosamente, descobrir um comportamento óptimo global. Em ambientes dinâmicos, o agente deve, na melhor das hipóteses, tentar continuar com as mudanças no ambiente e controlar constantemente o comportamento óptimo em mudança. As coisas são ainda mais complicadas nos Sistemas Multi-Agente, onde os agentes devem alterar de forma adaptável os ambientes de aprendizagem uns dos outros. A gama de ferramentas para modelar e analisar as dinâmicas dos aprendizes concorrentes é muito limitada. Muitas dessas ferramentas são particularmente apropriadas a apenas alguns métodos de aprendizagem, e poucas oferecem uma estrutura comum para múltiplas técnicas de aprendizagem, sendo a teoria de jogos evolucionários a principal ferramenta;

Modelação dos Membros da Equipa

Numa outra área em Aprendizagem Concorrente é a modelação dos membros da equipa, que consiste na aprendizagem de outros agentes no ambiente de forma que seja possível adivinhar os seus comportamentos esperados e agir de acordo com eles (por exemplo, para cooperar com eles mais efectivamente). Como os outros agentes se estão a modelar uns aos outros, isto trás o espectro da recursividade infinita: “o agente A está a fazer a acção X porque ele pensa que o agente B pensa que o agente A pensa que o agente B pensa que…”. É possível categorizar os agentes com base na complexidade que eles assumem para os outros membros da equipa. Um agente de “nível zero”

acredita que nenhum dos seus parceiros está a executar alguma actividade de aprendizagem e não considera os seus comportamentos em mudança como “adaptáveis” no seu modelo. Um agente de “nível um” modela os seus parceiros como agentes de “nível zero”. Em geral, um agente de “nível N” modela os outros membros da equipa como agentes de “nível (N-1)”. Quando se lida com um grande número de membros numa equipa, uma abordagem de modelação mais simples é presumir que a equipa inteira consiste em agentes idênticos ao agente modelado.

4.5. Sumário

Neste capítulo foram abordadas várias questões relacionadas com aprendizagem e sistemas baseados em agentes, aplicados ao problema do Escalonamento. Começou-se por descrever a base da aprendizagem computacional, a Aprendizagem Automática, com as diversas subáreas, desde a Aprendizagem Supervisionada, passando pela Aprendizagem Semi-Supervisionada, até à Aprendizagem Não-Supervisionada, definindo-se também a Aprendizagem por Reforço.

Seguidamente foi realizada uma descrição do Raciocínio baseado em Casos, uma vez que é a base do desenvolvimento do módulo de aprendizagem.

A questão principal deste capítulo, a Aprendizagem Multi-Agente em Escalonamento, foi abordada na secção 4.4. Primeiro foram descritos aspectos de aprendizagem em Sistemas de Fabrico baseados em agentes, seguidamente foi descrita a relação entre Aprendizagem Automática e Sistemas Multi-Agente e também alguns aspectos de aprendizagem de agentes. Foram também descritas algumas técnicas de aprendizagem, desde a Aprendizagem baseada em Lógica, a Aprendizagem Reactiva, a Aprendizagem Social, a Aprendizagem em Equipa, e a Aprendizagem Concorrente.

CAPÍTULO 5. O SISTEMA AUTODYNAGENTS E O MÓDULO DE AUTO-OPTIMIZAÇÃO

5.1. Introdução

O sistema AutoDynAgents (*Autonomic Agents with Self-Managing Capabilities for Dynamic Scheduling Support in a Cooperative Manufacturing System*) consiste num Sistema Multi-Agente para a resolução de problemas de Escalonamento sujeitos a perturbações. Este sistema utiliza técnicas de optimização para a obtenção de soluções quase-óptimas dos planos de escalonamento, nomeadamente Meta-heurísticas, tais como Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, Optimização por Colónia de Formigas, e *Particle Swarm Optimization*. A aplicação destas técnicas requer um conhecimento prévio da parametrização correcta das mesmas, adequadas ao problema a tratar. Esta especificação dos parâmetros correctos à Meta-heurística a usar é uma tarefa bastante difícil e requer alguma experiência e conhecimento pericial. Assim, será proposto neste capítulo um mecanismo de aprendizagem e auto-optimização para a parametrização automática das Meta-heurísticas a usar na resolução de problemas de Escalonamento.

Inicialmente será descrito o **sistema AutoDynAgents**, bem como toda a sua **arquitectura** e todos os tipos de agentes usados no Sistema Multi-Agente. Será também descrito o **Módulo de Escalonamento** e o **Módulo de Adaptação Dinâmica**. Finalmente será descrito o **Módulo de Auto-Optimização**, nomeadamente com a descrição da sua **arquitectura** interna e com a demonstração de um **exemplo ilustrativo**, para uma melhor percepção do seu funcionamento.

5.2. O sistema AutoDynAgents

O sistema AutoDynAgents (Madureira et al., 2008a)(Madureira et al., 2009b) é um sistema de escalonamento autónomo no qual uma comunidade de agentes modela um sistema real de fabrico sujeito a perturbações para a resolução autónoma, distribuída e cooperativa de problemas de escalonamento. É uma extensão das ideias concretizadas no sistema MASDScheGATS (Madureira et al., 2007) descrito anteriormente na secção 3.3.1.1.

A proposta do AutoDynAgents consiste num Sistema Multi-Agente onde existem agentes que representam tarefas (AgenteTarefa) e agentes que representam recursos/máquinas (AgenteRecurso). Cada AgenteRecurso deve ser capaz de encontrar uma solução local óptima ou quase-óptima através da aplicação de uma Meta-heurística implementada (Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, Optimização por Colónia de Formigas, ou *Particle Swarm Optimization*), de lidar com o dinamismo no sistema (ou seja, chegada de novas tarefas, cancelamento de tarefas, alteração dos atributos das tarefas, etc.), de escolher uma Meta-heurística e adaptar os seus parâmetros de

configuração de acordo com o problema a resolver, e comunicar com os outros agentes de modo a resolver o problema.

A abordagem de escalonamento seguida pelo sistema AutoDynAgents é um pouco diferente daquelas que são encontradas na literatura, uma vez que neste sistema cada AgenteRecurso é responsável por optimizar o escalonamento de operações para a máquina respectiva, através do uso de uma Meta-heurística. Esta abordagem considera um tipo específico de interacção social, a Resolução Cooperativa de Problemas, uma vez que um grupo de agentes trabalha conjuntamente para alcançar uma boa solução para o problema.

Nesta secção será descrito o sistema AutoDynAgents, nomeadamente a sua arquitectura, módulos e métodos principais.

5.2.1. Arquitectura

O sistema AutoDynAgents dispõe de uma arquitectura autónoma com um modelo em equipa e consiste num sistema de escalonamento, para a resolução de problemas Job-Shop Alargado dinâmicos, com dois componentes principais:

1. O módulo de escalonamento híbrido que usa Meta-heurísticas e um mecanismo para coordenação da actividade inter-máquina. O objectivo deste mecanismo é coordenar a operação das máquinas, tendo em consideração as restrições tecnológicas das tarefas, isto é, os relacionamentos 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.

O problema de escalonamento original é decomposto em séries de problemas de máquina única. Como já referido, os agentes máquina (AgenteRecurso) têm Meta-heurísticas associadas e obtêm soluções localmente para mais tarde dar origem a uma boa solução final viável para o problema.

Na Figura 13 e na Figura 14 será apresentada a arquitectura do sistema AutoDynAgents. Enquanto a primeira apresenta uma arquitectura global dos vários módulos e sua interligação, a segunda apresenta como os agentes estão interligados entre si. Como se pode ver, além do módulo de Interface Gráfica, existem seis tipos de agentes: AgenteUI, AgenteTarefa, AgenteRecurso, e os agentes Auto-* (Auto-Configuração, Auto-Optimização, e Auto-Recuperação).

Da análise da Figura 13 é possível verificar a interligação dos vários módulos. O módulo de Interface Gráfica é responsável pela definição do problema e parametrização base das Meta-heurísticas. O módulo de Auto-Configuração é responsável por detectar alterações no problema

(como chegada ou eliminação de tarefas, por exemplo) quer por definição na Interface Gráfica quer por eventos aleatórios.

Os eventos aleatórios são detectados pelo módulo de Auto-Configuração, e seguidamente comunicados ao sistema para adaptação dos parâmetros das Meta-heurísticas (através do módulo de Auto-Optimização) e adaptação do problema (através do módulo de Adaptação Dinâmica).

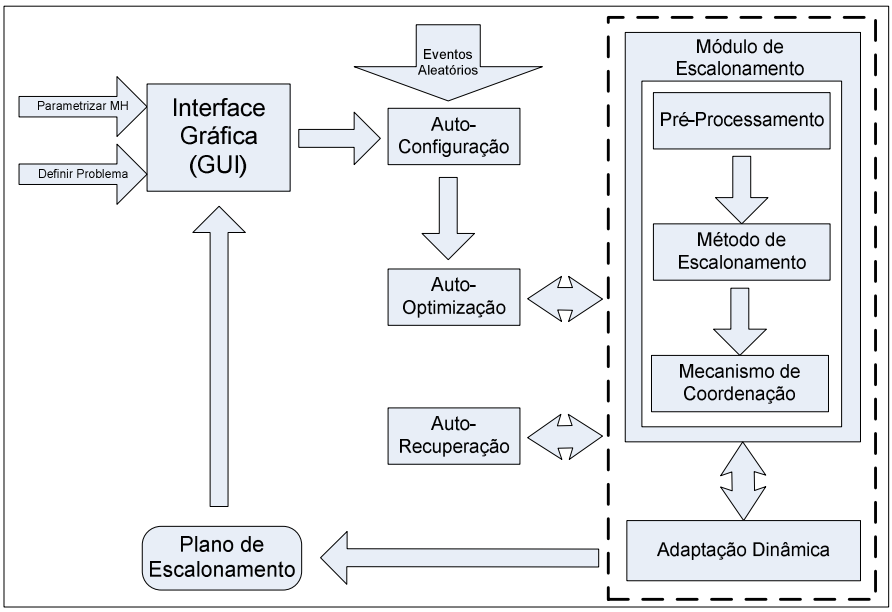

Figura 13 – Arquitectura global do sistema AutoDynAgents

O módulo de Auto-Optimização é responsável por escolher uma Meta-heurística a aplicar ao problema e respectiva parametrização, sendo o módulo de Auto-Reparação responsável por detectar falhas no Sistema Multi-Agente e respectiva correcção.

O Sistema Multi-Agente é constituído pelo Módulo de Escalonamento (subsecção 5.2.2) e pelo Módulo de Adaptação Dinâmica (subsecção 5.2.3), e gera um plano de escalonamento que é apresentado na Interface Gráfica.

Da análise da Figura 14 é possível verificar a interligação dos vários agentes presentes na arquitectura, com destaque para a comunicação entre os agentes do Sistema Multi-Agente. Assim, à semelhança da figura anterior, o agente de Auto-Configuração detecta as alterações ao problema provenientes do ambiente externo (Interface Gráfica ou eventos aleatórios) e comunica-os ao agente de Auto-Optimização, responsável, como já referido, pela definição dos parâmetros de uma Meta-heurística para a resolução do problema. Este por sua vez comunica com o AgenteUI, responsável pela criação dos vários AgentesTarefa, pela coordenação das soluções locais dos AgentesRecurso, e também pela comunicação da solução final ao agente de Auto-Optimização bem como à Interface Gráfica, para a respectiva visualização.

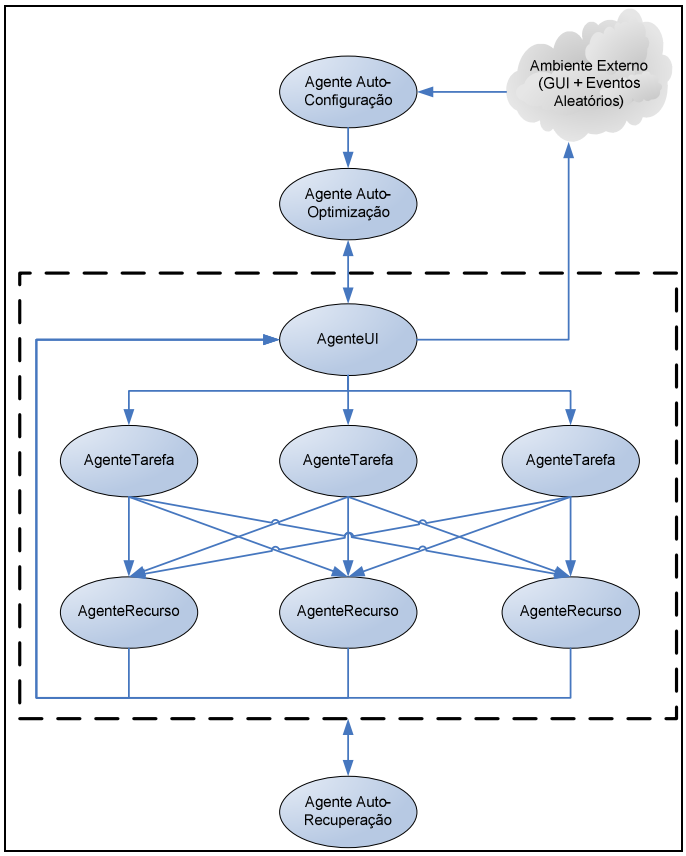

Figura 14 – Arquitectura de agentes do sistema AutoDynAgents

Cada AgenteTarefa efectua o pré-processamento da respectiva tarefa distribui as soluções pelos vários AgenteRecurso, cada um responsável por obter uma solução local para o problema de máquina única respectivo e comunicação ao AgenteUI.

Por último, o agente de Auto-Reparação encontra-se ligado a todos os agentes do Sistema Multi-Agente para detecção e correcção de possíveis falhas.

Seguidamente serão descritos com mais detalhe os vários módulos e comportamentos dos agentes presentes na arquitectura.

5.2.1.1. Interface Gráfica

O módulo da Interface Gráfica é responsável pela definição dos dados do problema e parametrização base das várias Meta-heurísticas presentes no sistema, assim como a visualização dos resultados obtidos.

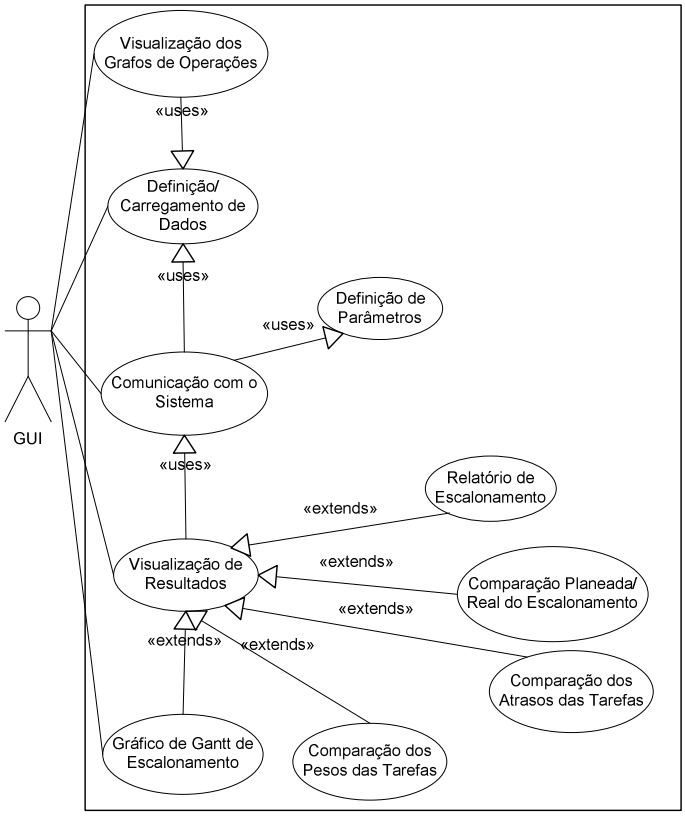

Figura 15 – Diagrama *Use-Case* da Interface Gráfica

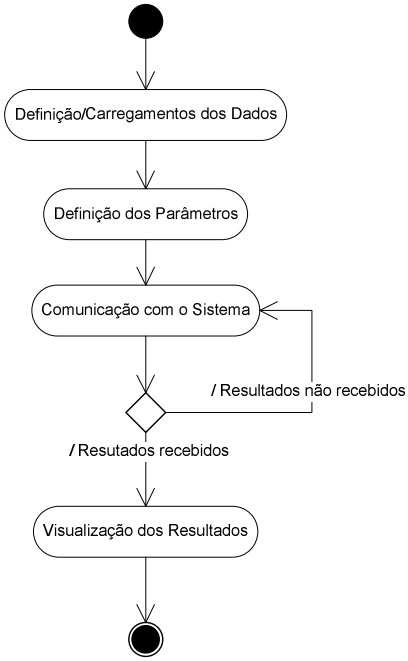

Figura 16 – Diagrama de Actividade da Interface Gráfica

Por análise do diagrama *Use-Case* da Figura 15 é possível detectar as seguintes funcionalidades:

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

Na Figura 16 está descrito o Diagrama de Actividades da Interface Gráfica. O primeiro passo é definir os dados de entrada. Caso esses dados estejam gravados em ficheiro XML, basta efectuar o carregamento do mesmo. Seguidamente é necessário definir a parametrização base pretendida, definindo os parâmetros para as Meta-heurísticas. Depois é iniciada a comunicação com o Sistema, neste caso com o AgenteUI.

Enquanto o processamento não termina, é possível continuar a trabalhar na Interface Gráfica, sendo possível, por exemplo, ver o grafo de precedências das operações de cada tarefa, tal como está exemplificado na Figura 17.

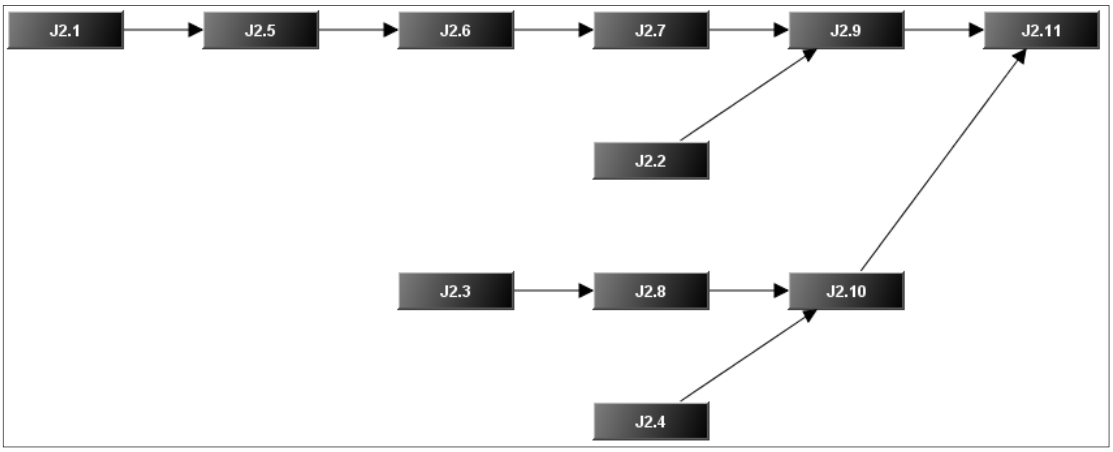

Figura 17 – Grafo de operações

Uma vez terminado o processamento da solução, é possível visualizar os resultados obtidos. Pode-se ver o gráfico de Gantt do escalonamento (Figura 19), gráficos comparativos (um de pesos das tarefas – Figura 18, outro de atrasos nas tarefas – Figura 20, e um de comparação entre o plano previsto e o elaborado – Figura 18) e o relatório de escalonamento (Figura 21).

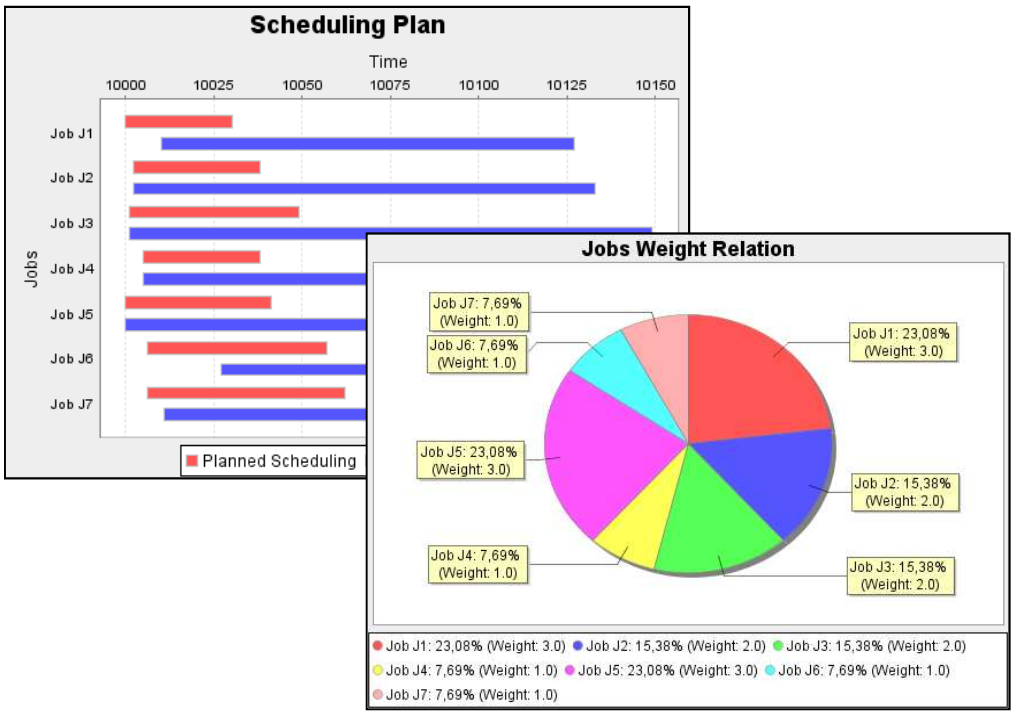

Figura 18 – Gráfico comparativo entre o plano previsto e o gerado, e gráfico comparativo dos pesos das tarefas

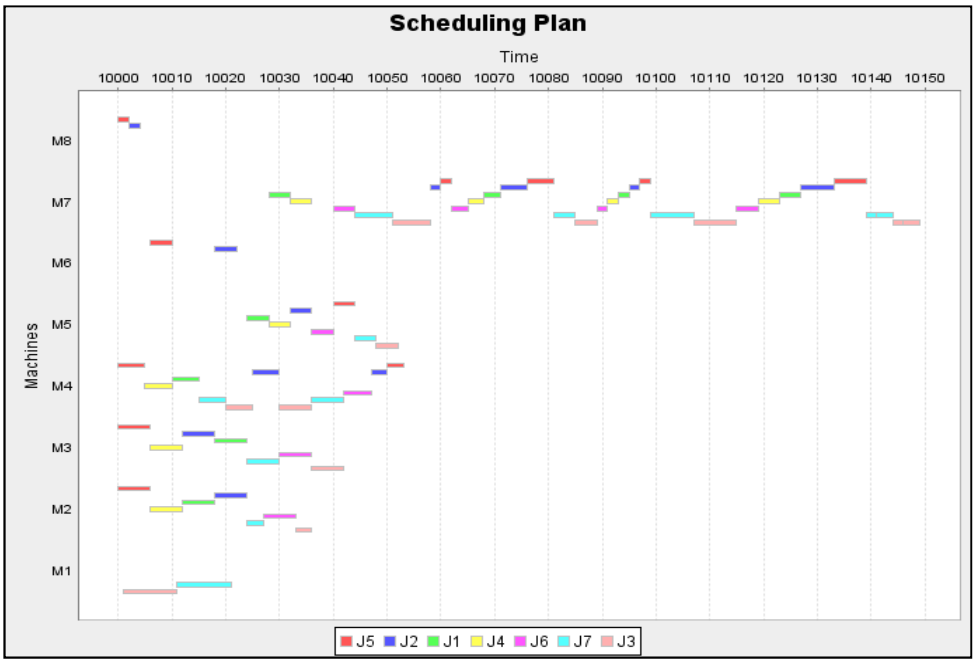

Figura 19 – Gráfico de Gantt do escalonamento

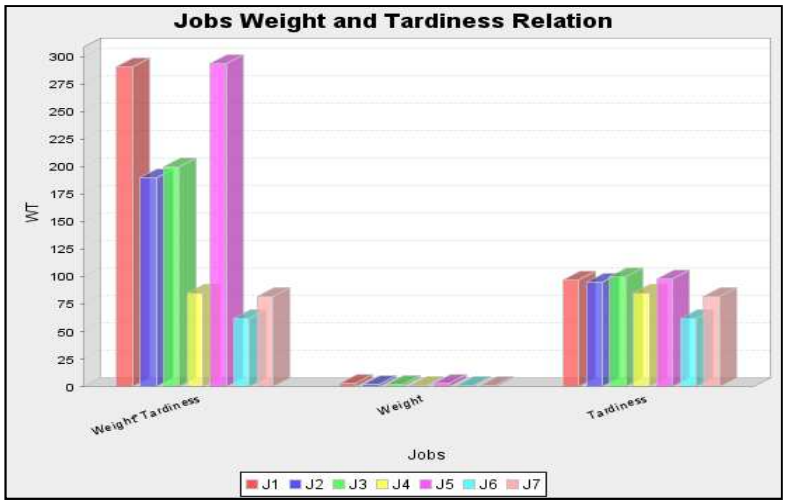

Figura 20 – Gráfico comparativo dos atrasos das tarefas

Job	Weight	Rel. Date	Due Date	Start	End	Fj	T	WT
J1	3	0	30	10.0	127.0	117.0	97.0	291.0
J2	2	2	38	2.0	133.0	131.0	95.0	190.0
J3	2	1	49	1.0	149.0	148.0	100.0	200.0
J4	1	5	38	5.0	123.0	118.0	85.0	85.0
J5	3	0	41	0.0	139.0	139.0	98.0	294.0
J6	1	6	57	27.0	119.0	92.0	62.0	62.0
J7	1	6	62	11.0	144.0	133.0	82.0	82.0

Summary:

Cmax	Tmax	N	Sum_Fj	Sum_Tj	Sum_...	Sum_...	Fmax	U	Time(s..
149.0	100.0	7.0	878.0	619.0	1748.0	1204.0	148.0	73.66	106.17

Figura 21 – Relatório de escalonamento

Também é possível introduzir alguns eventos após a recepção dos dados, como, por exemplo, inserção de novas tarefas, cancelamento de tarefas e alteração dos atributos das tarefas. Na definição da parametrização é possível indicar ainda a estratégia de reescalonamento a ser seguido, isto é, se o sistema aproveita o plano gerado anteriormente ou se recomeça o escalonamento ignorando o plano anterior, para lidar com o dinamismo.

5.2.1.2. AgenteUI

O AgenteUI é responsável pela comunicação do Sistema Multi-Agente com o meio exterior e o agente de Auto-Optimização, sendo também responsável pela coordenação das soluções enviadas pelos AgentesRecurso.

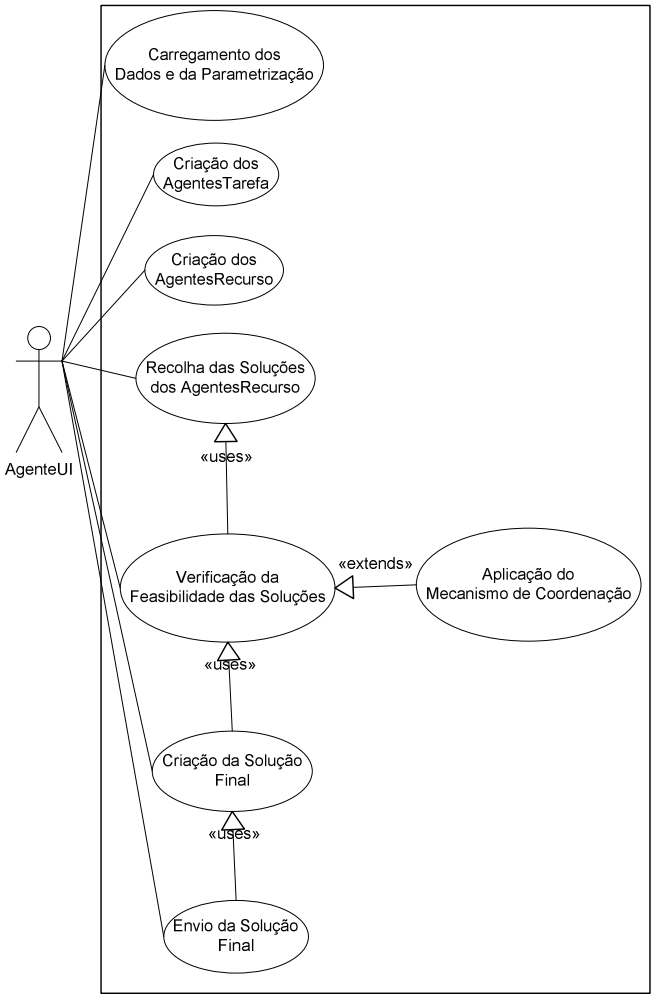

Figura 22 – Diagrama *Use-Case* do AgenteUI

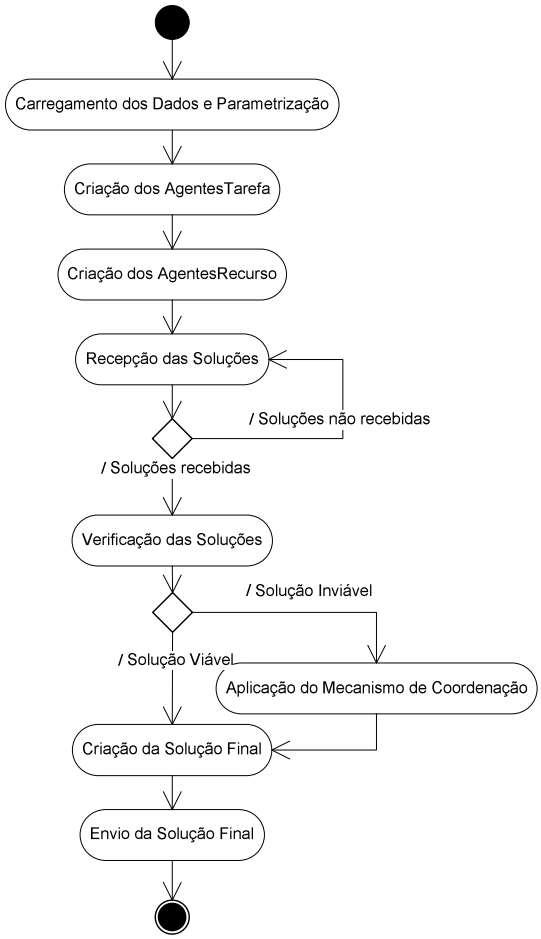

Figura 23 – Diagrama de Actividades do AgenteUI

Por análise do diagrama *Use-Case* da Figura 22 é possível identificar as seguintes funcionalidades:

- 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-Optimização);
- Criação dinâmica dos AgentesTarefa (cada agente corresponde a cada tarefa inserida no sistema);
- Criação dinâmica dos AgentesRecurso (cada agente corresponde a um recurso do sistema);
- Recolha das soluções dos vários AgentesRecurso;
- Verificação da exequibilidade das soluções;
- Aplicação do Mecanismo de Coordenação no caso de as soluções não serem exequíveis;
- Criação da solução final exequível e respectiva comunicação ao meio exterior e agente de Auto-Optimização.

O Diagrama de Actividades do AgenteUI está representado na Figura 23. A primeira coisa a fazer pelo AgenteUI é a leitura dos dados das tarefas e dos dados da parametrização da Meta-heurística a usar, provenientes do agente de Auto-Optimização.

Depois destes dados carregados, o AgenteUI procede à criação dos AgentesTarefa, um para cada tarefa definida, aos quais são enviados os dados e que serão responsáveis pelo pré-processamento das tarefas. Seguidamente são criados os AgentesRecurso, um para cada recurso (máquina) existente. Estes agentes irão calcular a solução para as operações na máquina respectiva.

Depois dos vários agentes criados, o AgenteUI fica à espera de receber as soluções dos AgentesRecurso. Só quando recebe todas as soluções é que avança para a verificação das mesmas. Se o conjunto das soluções formar uma solução não exequível, é aplicado um Mecanismo de Coordenação que coordena as relações de precedência entre as operações das tarefas com as relações de precedência nas máquinas, redefinindo os tempos de início e de conclusão para cada operação.

Depois de ser obtida uma solução final exequível, o AgenteUI cria um ficheiro XML com os dados finais para ser enviado à Interface Gráfica e ao agente de Auto-Optimização. Neste ficheiro são incluídos os dados finais de escalonamento das operações nas máquinas e um relatório final de escalonamento.

Devido à possibilidade de ocorrência de dinamismo, o AgenteUI não termina o seu processamento, ficando à espera de chegada de eventos. No caso de chegada de algum evento, o AgenteUI recomeça o processamento do início, mas só cria os AgentesTarefa e AgentesRecurso no caso de haver algum novo para criar e caso a opção da parametrização seja aproveitar o plano anteriormente gerado. No caso de a estratégia ser para recomeçar o escalonamento, então o plano anteriormente gerado é ignorado.

5.2.1.3. AgenteTarefa

Os AgentesTarefa são criados dinamicamente e são únicos por cada tarefa a processar pelo sistema. Cada AgenteTarefa é responsável pelo pré-processamento das operações que constituem a tarefa e pela distribuição das operações pelos AgentesRecurso.

No diagrama *Use-Case* da Figura 24 é possível identificar as seguintes funcionalidades:

- Recepção 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 respectivas características pelos respectivos AgentesRecurso.

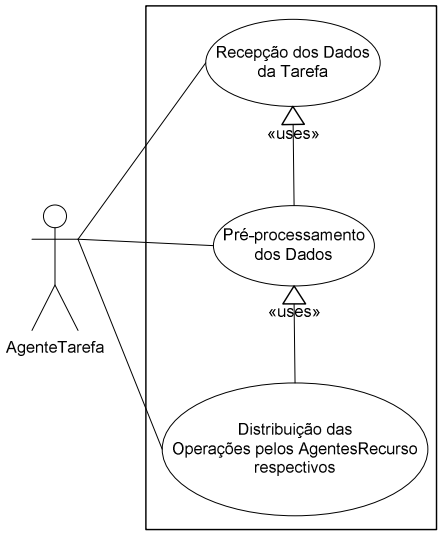

Figura 24 – Diagrama *Use-Case* dos AgentesTarefa

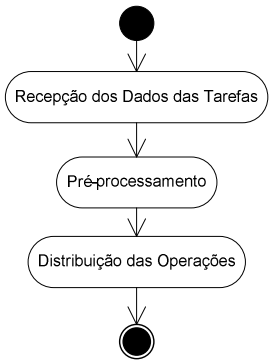

Figura 25 – Diagrama de Actividades dos AgentesTarefa

O Diagrama de Actividades dos AgentesTarefa está representado na Figura 25. Quando os AgentesTarefa são criados, estes recebem os dados das tarefas como parâmetro. Assim, a primeira acção a realizar é ler os dados das tarefas para seguidamente ser feito o pré-processamento das mesmas. Neste pré-processamento são calculados os tempos de início e de conclusão estimados das operações de cada tarefa. Depois de efectuado o pré-processamento, o AgenteTarefa distribui as operações pelos AgentesRecurso respectivos.

No caso de ocorrência de dinamismo, os AgentesTarefa não terminam o seu processamento após a distribuição das operações. Podem ocorrer alterações nos dados das tarefas que necessitem a re-execução do pré-processamento. No caso de o evento ser de eliminação de uma tarefa o AgenteTarefa respectivo termina o processamento, comunicando aos AgentesRecurso o sucedido.

Quando o novo evento se refere à chegada de tarefas, então novos AgentesTarefa são criados pelo AgenteUI.

5.2.1.4. AgenteRecurso

Os AgentesRecurso representam os recursos existentes no sistema para processar as operações constituintes das tarefas. Existe um por cada recurso da planta fabril e cada um é responsável pela descoberta da melhor solução de escalonamento das operações recebidas pelos diversos AgentesTarefa.

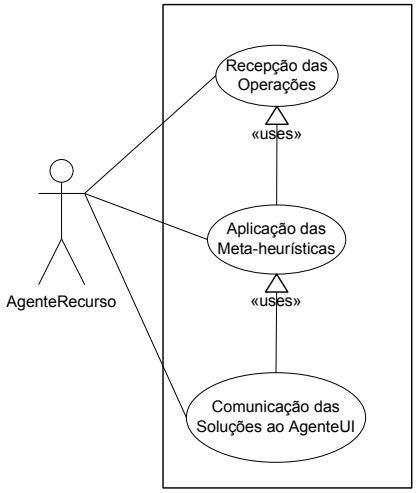

Figura 26 – Diagrama *Use-Case* dos AgentesRecurso

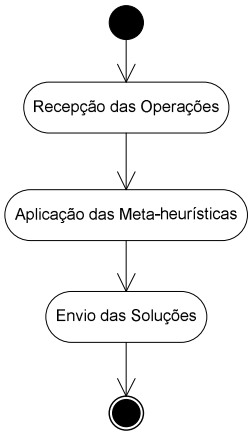

Figura 27 – Diagrama de Actividades dos AgentesRecurso

Por análise do diagrama *Use-Case* da Figura 26 é possível identificar as seguintes funcionalidades:

- Recepção das operações a serem processadas dos diversos AgentesTarefa;
- Aplicação de uma Meta-heurística, previamente escolhida e parametrizada pelo agente de Auto-Optimização, para encontrar a melhor solução possível para cada problema de máquina única;
- Comunicação da solução encontrada ao AgenteUI.

Na Figura 27 está descrito o Diagrama de Actividades dos AgentesRecurso.

Mal os AgentesRecurso sejam criados, estes ficam à espera de receber as operações pré-processadas pelos AgentesTarefa. Depois de as operações serem recebidas, os AgentesRecurso procedem ao cálculo das soluções através da aplicação de uma Meta-heurística dependendo da parametrização previamente definida.

Quando termina o cálculo das soluções, os AgentesRecurso enviam-nas para o AgenteUI que tratará de as incluir numa solução única.

Também 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.

5.2.1.5. Agentes Auto-*

Respectivamente aos agentes Auto-*, estes existem para que o conceito de Computação Autónoma seja aplicável ao sistema. Estas capacidades de auto-gestão são globais ao sistema, sendo que não existe um módulo único para cada agente do Sistema Multi-Agente. Assim, cada agente do sistema tem o seu módulo de Auto-Configuração, Auto-Optimização e Auto-Recuperação através de comunicações/monitorizações do respectivo agente de auto-gestão. A capacidade de Auto-Protecção não foi considerada.

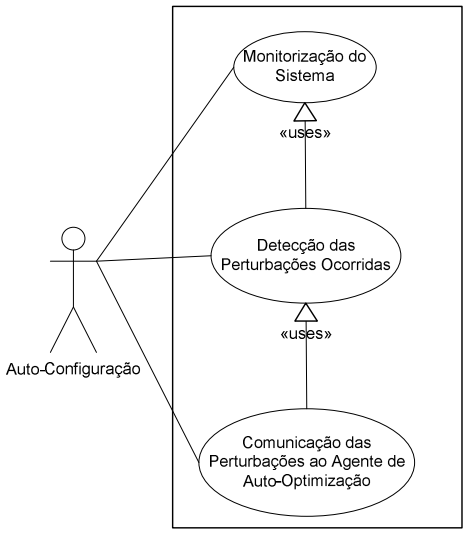

Figura 28 – Diagrama *Use-Case* do agente de Auto-Configuração

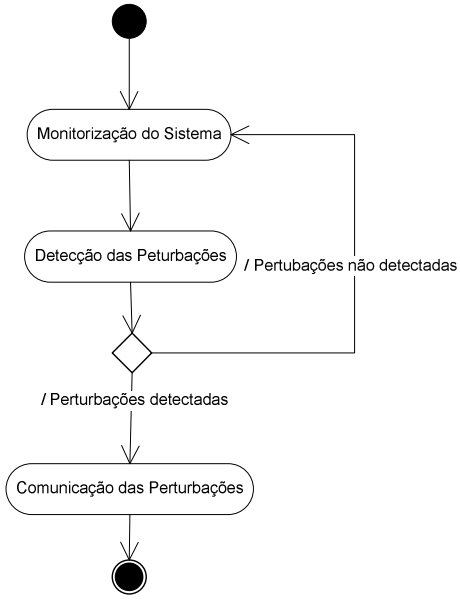

Figura 29 – Diagrama de Actividades do agente de Auto-Configuração

O agente de **Auto-Configuração** é responsável por monitorizar o sistema de modo a detectar 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 5.2.3.

Como já referido anteriormente, e por análise da Figura 28 e Figura 29, o agente de Auto-Configuração tem as seguintes funcionalidades:

- Monitorização contínua do sistema;
- Detecção das perturbações ocorridas;
- Comunicação das perturbações detectadas ao Agente de Auto-Optimização.

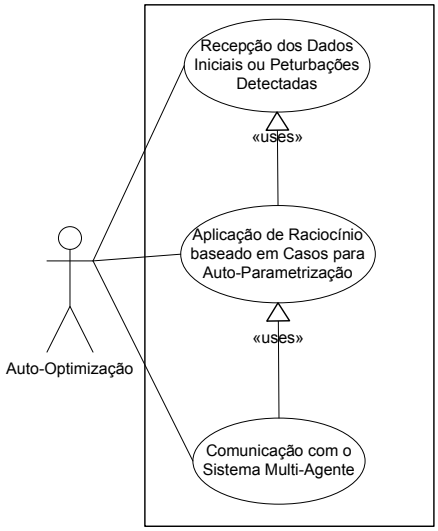

Figura 30 – Diagrama *Use-Case* do agente de Auto-Optimização

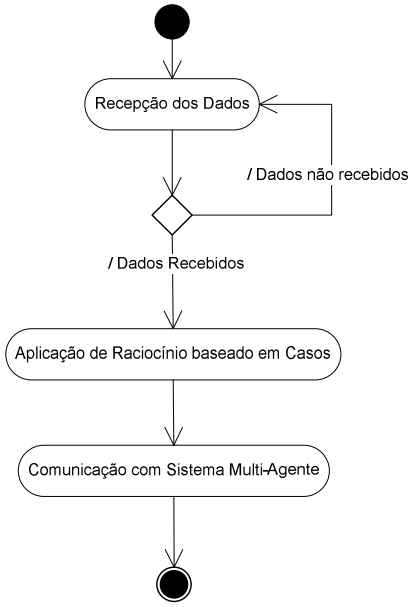

Figura 31 – Diagrama de Actividades do agente de Auto-Optimização

O agente de **Auto-Optimização** é responsável pela parametrização automática dos parâmetros das Meta-heurísticas, de acordo com o problema. Como se pode ver na Figura 30 e na Figura 31, este agente recebe o problema inicial, ou as mudanças detectadas pelo agente de Auto-Configuração, escolhe automaticamente a Meta-heurística a usar, bem como faz a sua auto-parametrização, e comunica com o Sistema Multi-Agente. Se ocorre algum tipo de dinamismo, os parâmetros podem ser alterados em tempo de execução. Esta auto-parametrização 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 respectivos parâmetros a usar. Quando um novo caso é resolvido, este é armazenado para uso futuro. Este módulo de Auto-Optimização encontra-se descrito detalhadamente na secção 5.3, uma vez que se refere ao objectivo central desta tese.

Finalmente, o agente de **Auto-Reparação** dá ao sistema a capacidade de diagnosticar desvios às condições normais e toma acções proactivamente de modo a normalizar o sistema e evitar interrupções de serviço. Este agente monitoriza os outros agentes de modo a fornecer capacidades de auto-reparação a todos eles (Figura 32 e Figura 33). Uma vez que os agentes podem falhar por qualquer razão, esta auto-reparação permite que os agentes efectuem cópias de segurança dos seus

registos, de modo ser a possível a sua reactivação para não se perder dados importantes. Os agentes podem ser restaurados a partir de um ponto anterior em vez de ser feito um *reset* total. Com este agente, o sistema torna-se estável, mesmo se ocorrer algum *deadlock* ou falha nos agentes do sistema.

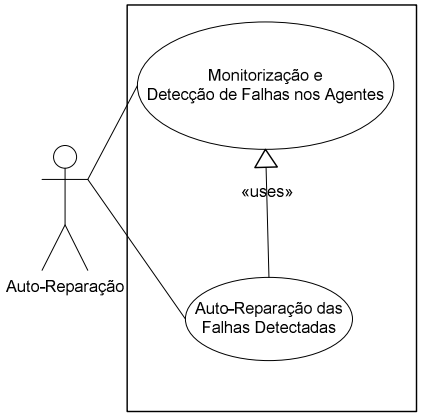

Figura 32 – Diagrama de *Use-Case* do agente de Auto-Reparação

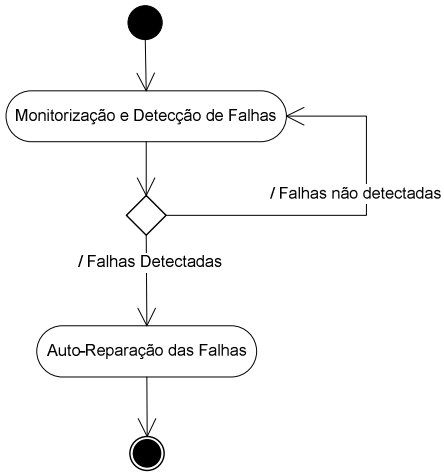

Figura 33 – Diagrama de Actividades do agente de Auto-Reparação

5.2.2. Módulo de Escalonamento

O módulo de Escalonamento é basicamente constituído pelo Método de Escalonamento baseado em Meta-heurísticas (Madureira, 2003).

Tabela 4 – Notação

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 efectivo (*tardiness)* da tarefa ($T_j = \max \{ L_j , 0 \}$)

Na Tabela 4 é apresentada a notação usada na descrição dos módulos e métodos.

As soluções são codificadas através de representação directa, onde o plano de escalonamento é descrito como uma sequência de operações, isto é, cada posição representa um índice de operação com tempos de processamento iniciais e finais. Cada operação é caracterizada pelo índice (*i*, *j*, *k*, *l*), onde *i* define a máquina onde a operação *k* é processada, a tarefa *j* a que pertence, e o nível *l* no grafo de precedências de operações (nível 1 corresponde às operações iniciais, sem precedentes).

Com a informação do problema é realizado um pré-processamento inicial, por parte de cada AgenteTarefa, que consiste na definição dos tempos de início e de conclusão das operações de cada tarefa do problema inicial determinístico.

O Método de Escalonamento baseado em Meta-heurísticas, como descrito na Tabela 5, é implementado em duas fases: a primeira tem por finalidade obter um plano para o problema *Job-Shop Alargado*, baseado na decomposição do problema original nos vários problemas de máquina única (AgentesRecurso) que o constituem. Nesta fase são usados diferentes mecanismos para a definição do intervalo dos tempos estimados de início e dos tempos de conclusão das operações, e heurísticas genéricas para a parametrização das Meta-heurísticas usadas. De seguida é aplicada uma Meta-heurística a cada um dos problemas de máquina única, e integradas as soluções obtidas no problema principal. A primeira fase tem já como resultado um plano de escalonamento.

Na segunda fase é verificada a exequibilidade do plano obtido e aplicado um Mecanismo de Coordenação quando necessário, para a coordenação dos tempos de ocupação das máquinas e das relações de precedência definidas para as operações.

Tabela 5 – 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 respectivas 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 aplicar o mecanismo de coordenação se necessário.**
Passo 6	Verificar se estamos perante uma solução válida ou exequível. Se não for válida então é necessário aplicar o Mecanismo de Coordenação.

Inicialmente, o problema *Job-Shop* Alargado determinístico é decomposto em problemas de máquina única. Assume-se a existência de tempos 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 AgenteRecurso respectivo, pela 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.

O tempo de conclusão para cada operação é derivado das datas de entrega das tarefas respectivas e do tempo de processamento através da subtracção do tempo de processamento ao tempo de conclusão da tarefa imediatamente seguinte (equação (6)).

$$C_{ijk-1l} = C_{ijkl} - p_{ijkl} \quad (6)$$

Este procedimento começa com a última operação da tarefa e acaba com a primeira. Quando uma operação é precedida por mais do que uma operação, isto é, quando existe uma estrutura multi-nível, o tempo de conclusão é o valor mais baixo, como definido na equação (7).

$$C_{ijkl-1} = \min\{C_{ijkl} - p_{ijkl}\} \forall\ l > l-1 \quad (7)$$

Os intervalos de tempo de início das operações $[t_{ijk}, T_{ijk}]$ são também definidos considerando os tempos de lançamento das tarefas e os tempos de processamento das operações. O tempo de início mais cedo t_{ijk} corresponde ao instante de tempo a partir do qual o processamento da operação pode ter início. O tempo de início mais tarde T_{ijk} 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 $[t_{ijk}, T_{ijk}]$ é a intercepção dos intervalos das operações precedentes correlacionados pelo tempos de processamento respectivos. Nesta fase, apenas são consideradas as restrições de precedência tecnológica das operações e 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 (8).

$$ITI_{ijkl} = [r_{ijkl}, C_{ijkl} - p_{ijkl}] \quad (8)$$

O intervalo de tempo de início para operações com um precedente é definido na equação (9).

$$ITI_{ijkl} = [t_{ijkl-1}, T_{ijkl-1}] + p_{ijkl-1} \quad (9)$$

O intervalo de tempo de início para operações com mais do que um precedente é a intercepção dos intervalos de tempo de início de todas as operações precedentes correlacionadas pelos respectivos tempos de processamento (equação (10)).

$$ITI_{ijkl} = [t_{ijK1L}+p_{ijK1L}, T_{ijK1L}+p_{ijK1L}] \cap [t_{ijK2L}+p_{ijK2L}, T_{ijK2L}+p_{ijK2L}] \cap \ldots \cap [t_{ijKnL}+p_{ijKnL}, T_{ijKnL}+p_{ijKnL}] \quad (10)$$
$$\text{com } Kn<k \text{ e } L<l$$

Seguidamente são descritos com mais detalhe o Mecanismo de Coordenação e o Mecanismo de Geração de Vizinhanças usado nas Meta-heurísticas, bem como detalhes sobre a implementação das várias Meta-heurísticas.

5.2.2.1. Mecanismo de Coordenação

A integração das soluções para os vários problemas de máquina única pode levar a planos inviáveis para o problema de escalonamento. Tal acontece por se considerar apenas as relações de precedência nas máquinas e as datas de conclusão das tarefas, em cada máquina única. A actividade 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.*

Tabela 6 – Mecanismo de Coordenação

Passo 1	Coordenar a actividade das máquinas começando pelas operações do primeiro nível. Estas não têm operações precedentes. Neste nível, os instantes de início e de conclusão das operações mantêm-se, isto é, 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 processo (*Passo 2*) repete-se até todas as operações terem sido escalonadas.

O Mecanismo de Coordenação descrito na Tabela 6 tem por objectivo 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.

O plano resultante é constituído por um conjunto de sequências de operações, uma por cada máquina. O processo de coordenação consiste em percorrer todas as sequências e tentar "coordenar" a actividade 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.

5.2.2.2. Implementação das Meta-heurísticas

Seguidamente serão 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). Como o problema de escalonamento é baseado em permutações de tarefas, faz sentido definir um afastamento máximo para que se possa restringir algumas soluções potencialmente más. 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 (por exemplo, com um afastamento máximo de 25% num problema de 20 tarefas, cada tarefa poderá trocar com um dos seus 5 vizinhos à direita e à esquerda).

Também foram consideradas algumas das regras de prioridade descritas na secção 2.4.1, para a geração da solução inicial antes da aplicação das Meta-heurísticas, nomeadamente a EDD, SPT, REDD, RND, e SeqNivel.

Pesquisa Tabu

A Pesquisa Tabu foi importada do sistema MASDScheGATS, cuja implementação foi desenvolvida com recurso a uma *framework* desenvolvida em Java por Robert Harder, designada OpenTS[7], que suporta a implementação desta Meta-heurística, duma forma orientada a objectos.

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

A *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 o facto de percorrer a vizinhança de soluções, seleccionar a melhor solução, e lidar com a lista tabu, duma maneira transparente para o programador.

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

Os parâmetros de entrada desta Meta-heurística são:

- **Critério de Paragem** – número de iterações que o algoritmo efectua;
- **Afastamento máximo** – percentagem de geração de vizinhança;
- **Subvizinhança** – percentagem de subvizinhança;
- **Lista Tabu** – tamanho da lista tabu.

Simulated Annealing

Esta Meta-heurística foi implementada directamente a partir do algoritmo original, descrito na subsecção 2.4.2.2. Os parâmetros de entrada do algoritmo são:

- **Critério de Paragem** – número de iterações que o algoritmo efectua;
- **Número de iterações *k*** – número de iterações à mesma temperatura.
- **Temperatura Inicial** – temperatura com que o algoritmo começa a processar;
- **Alpha** – representa o factor de redução de temperatura, sendo um valor 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 do normal. 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 aumentar ligeiramente a eficiência considerou-se uma selecção aleatória do índice do vizinho a seleccionar antes da geração da vizinhança, para que o método retorne quando encontrar o vizinho seleccionado, em vez de se gerar a totalidade da vizinhança e só depois seleccionar um vizinho aleatoriamente. Desta forma não é necessário guardar a vizinhança e apenas no pior caso é percorrida a totalidade da vizinhança, aumentando ligeiramente a eficiência. De modo a ser possível saber o número total de vizinhos para ser seleccionado um aleatoriamente foi aplicada a seguinte fórmula:

$$\text{num\_vizinhos} = k(n-k) + \sum_{i=n-k+1}^{n-1} n - i \qquad (11)$$

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

Algoritmos Genéticos

Semelhante à Pesquisa Tabu, também esta Meta-heurística foi importada do sistema MASDScheGATS, e implementada com recurso a uma *framework* externa, neste caso a WATCHMAKER[8], desenvolvida por Daniel W. Dyer. Esta é uma *framework* orientada a objectos para Computação Evolucionária, desenvolvida igualmente em Java.

[8] https://watchmaker.dev.java.net/

Assim como na *framework* da Pesquisa Tabu, também esta requereu o desenvolvimento de algumas classes específicas ao problema de escalonamento, nomeadamente a classe para cálculo da função objectivo, a classe geradora de populações (onde se implementou o mecanismo de geração de populações descrito anteriormente), e uma classe para cada operador usado (cruzamento, mutação e selecção). Como operador de cruzamento foi implementado o *order crossover.*

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

Os parâmetros de entrada 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;
- **Taxa de cruzamento –** taxa de cruzamento de indivíduos;
- **Taxa de mutação –** taxa de mutação de indivíduos.

Optimização por Colónia de Formigas

Na implementação desta Meta-heurística tentou-se usar uma aplicação o mais directa possível do algoritmo original, descrito na subsecção 2.4.2.4, embora com algumas adaptações. Um caminho é 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 do feromona.

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

o respectivo valor objectivo, 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 campo 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.

A título de exemplo, num problema de 5 tarefas, considerem-se duas formigas, 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.

Na Tabela 7 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.

Tabela 7 – 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

Em cada iteração, quando as formigas constroem os caminhos, a escolha de cada ramo funciona como uma probabilidade, ou seja, poderá ser escolhido um ramo mau, 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), página 18) é calculado em tempo real de construção de um caminho, ou seja, vai sendo aplicada a função objectivo 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 objectivo poder variar consoante a parametrização escolhida.

Depois de um novo caminho estar construído, este é comparado com o caminho actual que a formiga está a percorrer e só é feita uma mudança de caminho se o novo caminho for melhor que o actual. Esta restrição foi colocada para não permitir que a formiga esteja, no pior caso, sempre a mudar de caminho, impedindo a estagnação da colónia, pois também não faz sentido uma formiga

estar a percorrer um bom caminho e mudar para um pior. E com base nos testes efectuados, 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.

Particle Swarm Optimization

À semelhança das outras Meta-heurísticas, também a *Particle Swarm Optimization* foi implementada tentando ser o mais fiel possível ao algoritmo inicial, no entanto com 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 (por exemplo, referencial de 3 dimensões corresponde a um referencial xyz).

Sendo assim, o algoritmo é constituído por partículas, cada uma com as suas dimensões, por uma solução óptima encontrada (óptimo local), e por um valor de *fitness* actual da partícula. Considerando que esse conjunto de partículas diz respeito a uma população, existe uma forma para guardar a melhor solução encontrada desde o início de funcionamento do algoritmo (solução óptima 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 confuso. 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 efectuados para cada uma dessas dimensões de cada partícula. O seu valor de *fitness* é calculado após ordenação das dimensões por ordem crescente (minimização), e guardado no *fitness* corrente de cada uma. O valor do óptimo 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 do óptimo global, como descrito na literatura, é obtido após apuramento da solução com mais qualidade encontrada entre os óptimos locais. Foi preciso ter em atenção que

quando as dimensões de cada partícula são ordenadas para cálculo do *fitness*, essa ordenação é temporária, efectuada apenas para esse cálculo.

Utilizou-se uma classe responsável por armazenar as informações referentes a cada partícula, nomeadamente o óptimo local, o *fitness*, o *fitness* óptimo, 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 velocidade e 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 de entrada desta Meta-heurística são:

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

5.2.3. Módulo de Adaptação Dinâmica

Em problemas não-determinísticos, alguns parâmetros são incertos, isto é, não são fixos como se assume nos problemas determinísticos. O não-determinísmo 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[9], e depois, se ocorrer alguma perturbação durante a execução, o plano de escalonamento pode ser alterado ou revisto convenientemente, ou seja, é efectuado 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, então 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 actual tendo em consideração o tipo de perturbações ocorridas.

[9] Plano global de escalonamento gerado com antecedência.

Os tipos de eventos que requerem reescalonamento são:

- **Eventos parciais**, que implicam alterações nos atributos das tarefas ou operações, tais como tempos de processamento, datas de entrega, datas de lançamento e 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, isto é, 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 actual. O Mecanismo de Integração analisa o grafo de precedências de operações e, de acordo com a regra respectiva, introduz cada operação no problema de máquina única respectivo.

No sistema AutoDynAgents são considerados os seguintes Mecanismos de Integração:

- **Aleatório (RND)** – consiste em seleccionar aleatoriamente uma posição e inserir a nova tarefa nessa posição na solução actual 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 a posição da tarefa correspondente seja eliminada da solução actual ou em todos os cromossomas.

Depois de resolvido o problema de integração ou eliminação de tarefas é necessário implementar um **Mecanismo de Regeneração**. Este mecanismo tem por finalidade restaurar a vizinhança ou população modificada, por aplicação de mecanismos e 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, já descritos anteriormente.

A eliminação de uma tarefa requer a aplicação de um Mecanismo de Eliminação que retire ou elimine o gene ou a posição correspondente, para a eliminação das operações nos problemas das máquinas respectivas.

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 actual 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 actual e continuação do processo de pesquisa de melhores soluções.

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

5.3. Módulo de Auto-Optimização

As Meta-heurísticas tais como Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, etc., são bastante úteis na obtenção de boas soluções em tempos de execução viáveis, sendo que algumas vezes podem mesmo obter as soluções óptimas. Mas para que estas soluções óptimas ou quase-óptimas possam ser atingidas, é necessária a correcta parametrização desses algoritmos,

tarefa bastante difícil, que requer algum conhecimento pericial da Meta-heurística a usar e do problema a tratar. Algumas vezes é necessário usar o método de tentativa-erro para afinar os parâmetros, e a dificuldade aumenta quando existe a possibilidade de uso de mais do que uma Meta-heurística, pois nesse caso é requerida a escolha *a priori* da técnica a usar e só depois a parametrização da mesma.

Um dos objectivos do AutoDynAgents é que o sistema seja capaz de adoptar e fornecer capacidades de auto-parametrização para os métodos de resolução de problemas (Meta-heurísticas), de acordo com o problema a ser resolvido. Estes parâmetros podem ser alterados em tempo de execução, devido a dinamismo que possa entretanto ocorrer. O sistema deve então ser capaz de escolher uma Meta-heurística a usar e definir os seus parâmetros de entrada, de acordo com a situação actual. Além disso, pode ser possível comutar duma técnica para outra, dependendo do problema actual a tratar e da experiência passada acumulada. Esta auto-optimização é efectuada através de aprendizagem e experiência.

Para resolver este problema, foi proposto o desenvolvimento de um módulo incorporando ideias da Computação Autónoma, designado de **Auto-Optimização**, descrito nesta secção, e já iniciado anteriormente na subsecção 3.4.1.2 e na subsecção 5.2.1.5.

Fazendo um enquadramento nas técnicas de aprendizagem usadas em Sistemas Multi-Agente descritas na secção 4.4.4 e uma vez que, como já referido anteriormente, existe apenas um agente responsável pelo processo de aprendizagem, estamos perante uma Aprendizagem em Equipa, onde existe apenas um aprendiz com o objectivo de aprender e melhorar o desempenho da equipa (subsecção 4.4.4.3). Contrariamente a esta abordagem podia ser usada Aprendizagem Concorrente (subsecção 4.4.4.4), onde cada AgenteRecurso seria responsável por aprender e melhorar cada problema de máquina única. Assim, cada AgenteRecurso teria uma visão interior e apenas conseguiria melhorar o seu próprio desempenho, não sabendo nada sobre o desempenho global do sistema. Por isto mesmo, e uma vez que o objectivo é melhorar os planos globais do sistema, decidiu-se usar uma Aprendizagem em Equipa em oposição a uma Aprendizagem Concorrente.

O agente de Auto-Optimização é capaz de monitorizar o sistema e configurar os parâmetros das diferentes Meta-heurísticas, com respeito a cada problema que surge no sistema. Assim, este módulo tem de saber como deve parametrizar cada Meta-heurística, e, uma vez que é impossível prever todos os problemas a tratar, deve ser capaz de aprender com a experiência durante o seu tempo de vida como, por exemplo, os humanos. Para efectuar este mecanismo de aprendizagem, foi usado Raciocínio baseado em Casos, descrito anteriormente na secção 4.3.

Existem duas abordagens possíveis de aplicação do Raciocínio baseado em Casos no sistema. A **primeira** 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 respectivos parâmetros a usar. A **segunda** abordagem consiste em escolher primeiro a Meta-heurística a usar e só depois recuperar o caso mais similar, contendo os parâmetros a usar por essa Meta-heurística.

Considera-se a primeira abordagem mais adequada ao problema em questão uma vez que é importante que o sistema decida qual a Meta-heurística a usar e os respectivos parâmetros de configuração, pois nem todas as Meta-heurísticas são adequadas a qualquer tipo de problemas. Desta forma irá ser possível identificar as Meta-heurísticas que se adequam mais a um determinado tipo de problema.

Seguidamente serão descritos vários aspectos da implementação do módulo de Auto-Optimização, através da descrição da sua arquitectura, nomeadamente a base de casos e cada uma das fases do Raciocínio baseado em Casos, com ilustração de um exemplo para que seja perceptível o seu funcionamento.

5.3.1. Arquitectura

O módulo de Auto-Optimização consiste num sistema de Raciocínio baseado em Casos, cujo ciclo é igual ao descrito anteriormente na secção 4.3.5, composto por quatro processos principais, "os quatro R's", isto é, Recuperação, Reutilização, Revisão e Retenção. Este ciclo ilustrado na Figura 34 contém apenas uma adição na fase de Revisão. Esta adição passa por uma comunicação com o Sistema Multi-Agente, para ser possível executar o caso actual e retirar resultados relativos aos tempos de conclusão e de execução para se proceder à retenção do caso para uso futuro.

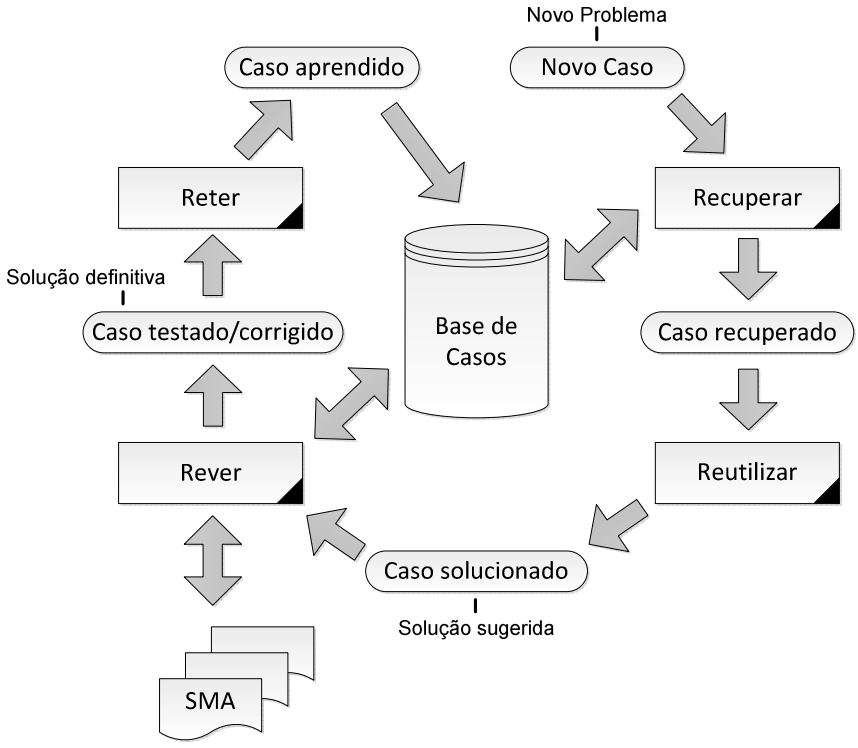

Figura 34 – Arquitectura do módulo de Raciocínio baseado em Casos

Assim, sempre que surge um novo problema para tratar, este corresponde a 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 Multi-Agente, é feita uma **revisão** da solução sugerida, através da adaptação e refinamento da mesma. Depois de ser efectuada a comunicação com o Sistema Multi-Agente, o caso testado é dado como uma solução definitiva, que por sua vez é **retida** na Base de Casos como um novo caso aprendido.

Seguidamente serão descritos detalhadamente os principais componentes da arquitectura, começando-se pela Base de Casos, seguindo-se as fases de Recuperação, Reutilização, Revisão e Retenção.

5.3.1.1. Base de Casos

A Base de Dados usada para armazenar os casos do módulo de Raciocínio baseado em Casos é composta por seis tabelas e encontra-se ilustrada na Figura 35.

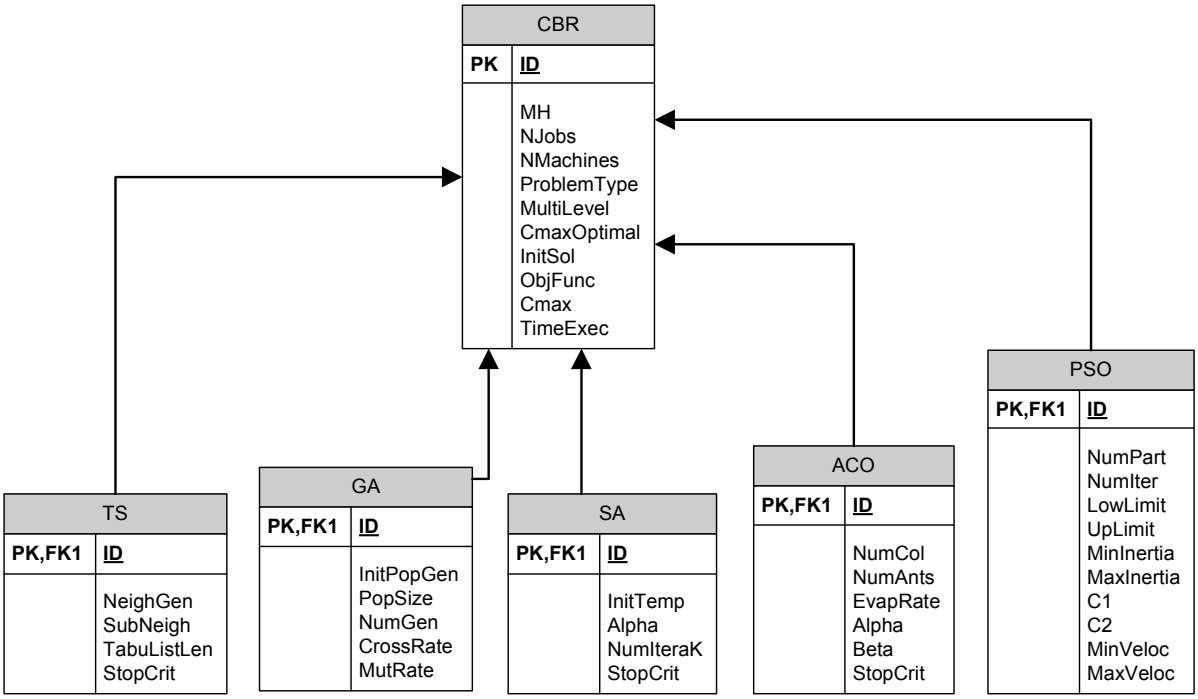

Figura 35 – Base de Casos do módulo de Raciocínio baseado em Casos

A tabela principal é a tabela CBR, uma vez que é responsável por armazenar os dados de todos os casos do sistema. De entre estes dados salienta-se os atributos presentes na medida de similaridade, descrita mais à frente na fase de Recuperação, nomeadamente o número de tarefas, o número de máquinas, o tipo de problema, a característica multi-nível, e o tempo de conclusão óptimo,

se conhecido. Esta tabela guarda também os tempos de conclusão e de execução obtidos por cada caso. Além disso, e tratando-se de uma tabela global às outras, guarda também a solução inicial e a função objectivo sugerida pelo Raciocínio baseado em Casos, atributos estes comuns a todas as tabelas. Na Tabela 8 encontra-se uma descrição de todos os campos, com especificação dos tipos de dados usados.

Tabela 8 – Descrição dos campos da tabela CBR

	Campo	Tipo de Dados	Definição
CBR	ID	Integer	Identificador único do caso, automaticamente incrementado.
	MH	String	Nome da tabela da Meta-heurística a utilizar (TS, GA, SA, ACO, ou PSO).
	NJobs	Integer	Número de tarefas do problema.
	NMachines	Integer	Número de máquinas do problema.
	ProblemType	String	Tipo de problema (*single-machine*, *Open-Shop*, *Flow-Shop* ou *Job-shop*).
	MultiLevel	Boolean	Indica se o problema é ou não um problema multi-nível, ou seja, se tem operações com mais do que um precedente.
	CmaxOptimal	Integer	Valor óptimo de tempo de conclusão (C_{max}) conhecido. Se não for conhecido então não é considerado.
	InitSol	String	Regra de Prioridade usada para construir a solução inicial (SeqNivel, EDD, SPT, REDD, ou RND).
	ObjFunc	String	Função objectivo usada para minimização do problema (C_{max}, WT, ou L_{max}).
	Cmax	Integer	Tempo de conclusão (C_{max}) obtido pelo caso.
	TimeExec	Double	Tempo computacional de execução do problema do caso.

As restantes tabelas correspondem às parametrizações/soluções para cada caso, ou seja, os parâmetros de entrada de cada Meta-heurística. Assim, existe uma tabela por cada Meta-heurística usada no sistema, isto é, Pesquisa Tabu (TS), Algoritmos Genéticos (GA), *Simulated Annealing* (SA), Optimização por Colónia de Formigas (ACO), e *Particle Swarm Optimization* (PSO), sendo os respectivos campos descritos na Tabela 9, Tabela 10, Tabela 11, Tabela 12, e Tabela 13, respectivamente.

Tabela 9 – Descrição dos campos da tabela TS

	Campo	Tipo de Dados	Definição
TS	StopCrit	Integer	Critério de paragem, número de iterações.
	NeighGen	Double	Percentagem de geração de vizinhança, vulgo afastamento máximo.
	SubNeigh	Double	Percentagem de subvizinhança.
	TabuListLen	Integer	Tamanho da lista tabu.

Tabela 10 – Descrição dos campos da tabela GA

	Campo	Tipo de Dados	Definição
GA	NumGen	Integer	Número de gerações.
	InitPopGen	Double	Percentagem de geração da população inicial, vulgo afastamento

			máximo.
	PopSize	Double	Percentagem do tamanho da população inicial.
	CrossRate	Double	Taxa de cruzamento.
	MutRate	Double	Taxa de mutação.

Tabela 11 – Descrição dos campos da tabela SA

	Campo	Tipo de Dados	Definição
SA	StopCrit	Integer	Critério de paragem, número de iterações.
	NumIteraK	Integer	Número de iterações à mesma temperatura.
	InitTemp	Double	Temperatura inicial.
	Alpha	Double	Factor *alpha* de redução da temperatura (arrefecimento).

Tabela 12 – Descrição dos campos da tabela ACO

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

Tabela 13 – Descrição dos campos da tabela PSO

	Campo	Tipo de Dados	Definição
PSO	NumItera	Integer	Número de iterações.
	NumPart	Integer	Número de partículas.
	MinVeloc	Double	Velocidade mínima.
	MaxVeloc	Double	Velocidade máxima.
	MinInertia	Double	Inércia mínima.
	MaxInertia	Double	Inércia máxima.
	C1	Double	Componente cognitivo.
	C2	Double	Componente social.
	LowLimit	Integer	Limite inferior.
	UpLimit	Integer	Limite superior.

5.3.1.2. Fase de Recuperação

A fase de Recuperação tem como objectivo pesquisar a Base de Casos, encontrar os casos passados mais similares com o novo caso, e recuperá-los para análise, de modo a posteriormente ser seleccionado um dos melhores casos e reutilizá-lo na próxima fase.

O algoritmo desta fase encontra-se descrito em pseudo-código na Tabela 14. O algoritmo tem dois parâmetros de entrada, a Base de Casos e o novo caso a tratar. Como parâmetro de saída, devolve uma lista de casos recuperados, em que cada elemento da lista corresponde a um par caso-similaridade, em que é especificado cada caso e respectiva similaridade com o novo caso.

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. Considerou-se esta similaridade mínima igual a 0.75, num intervalo [0 ; 1], o que significa que um caso necessita de ser 75% similar com o novo caso para ser seleccionado.

Depois da inicialização das variáveis, o algoritmo começa por efectuar uma pré-selecção de casos, cujo objectivo é tirar partido das vantagens de uma consulta à Base de Casos para pré-seleccionar apenas casos que possam ser considerados suficientemente similares, ou seja, para se ignorar casos que serão muito pouco similares com o novo caso. Esta pré-selecção é explicada mais à frente.

Tabela 14 – Algoritmo da fase de Recuperação

```
    Entrada: BaseCasos = {Caso1, Caso2, …, Cason}; NovoCaso
    Saída: ListaCasos // Lista de pares de casos recuperados e respectiva similaridade com o novo caso
1   ListaCasos ← {};
2   SimMin ← 0.75; // Similaridade mínima exigida para um caso ser seleccionado
3   // Efectuar uma pré-selecção de casos da Base de Casos
4   PreCasos ← Consultar(BaseCasos);
5   // Percorrer casos  pré-seleccionados
6   Para cada Casoi ∈ PreCasos Faz
7           // Calcular a medida de similaridade para cada um deles
8           Sim ← CalcularSimilaridade(Casoi, NovoCaso);
9           // Comparar com a similaridade mínima exigida
10          Se Sim>=SimMin Então
11                  // Adicionar à lista de casos
12                  ListaCasos.Adicionar(Casoi, Sim);
13          Fim
14  Fim
15  Devolver ListaCasos;
```

Depois da pré-selecção de casos da Base de Casos, é analisado cada um deles de modo a comparar a respectiva similaridade com a similaridade mínima exigida. Assim, para cada caso da lista de casos pré-seleccionados, é calculada a respectiva 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 a similaridade. No final do algoritmo, esta lista de casos é devolvida.

Tabela 15 – Comando SELECT de pré-selecção de casos da Base de Casos

```
SELECT c.*
FROM CBR c
WHERE (c.Njobs between MinNjobs and MaxNjobs) and (c.Nmachines between MinNmachines and MaxNmachines);
```

A pré-selecção de casos a partir da Base de Casos é efectuada através do comando SELECT descrito na Tabela 15. Com este comando, são seleccionados os casos que poderão ser candidatos à lista de casos a devolver. Assim, não serão seleccionados aqueles casos muito pouco similares com o novo caso, uma vez que a sua análise seria um desperdício computacional.

Neste comando são consideradas duas restrições sobre atributos, nomeadamente os atributos de número de tarefas e de número de máquinas do problema, pois são estes os atributos mais importantes, como se poderá constatar à frente na descrição da medida de similaridade. Cada um destes atributos deve estar entre dois valores, um valor mínimo e um valor máximo. Os limites do atributo de número de tarefas são descritos nas equações (12) e (13) respectivamente, e os limites do número de máquinas nas equações (14) e (15).

$$\mathrm{MinNjobs} = \mathrm{Njobs}_{\mathrm{NovoCaso}} * \mathrm{Peso}_{\mathrm{Njobs}} \tag{12}$$

$$\mathrm{MaxNjobs} = \frac{\mathrm{Njobs}_{\mathrm{NovoCaso}}}{\mathrm{Peso}_{\mathrm{Njobs}}} \tag{13}$$

$$\mathrm{MinNmachines} = \mathrm{Nmachines}_{\mathrm{NovoCaso}} * \mathrm{Peso}_{\mathrm{Nmachines}} \tag{14}$$

$$\mathrm{MaxNmachines} = \frac{\mathrm{Nmachines}_{\mathrm{NovoCaso}}}{\mathrm{Peso}_{\mathrm{Nmachines}}} \tag{15}$$

O cálculo dos valores mínimos é efectuado através da multiplicação do valor do atributo pelo peso respectivo. Estes pesos serão explicados mais à frente na descrição da medida de similaridade, sendo cada peso um número real do intervalo [0 ; 1]. Assim, o inverso é realizado para o cálculo dos valores máximos, ou seja, é efectuada a divisão dos valores dos atributos pelo respectivo peso.

Medida de Similaridade

Os atributos a considerar para a medida de similaridade são alguns campos da tabela CBR da Base de Casos, nomeadamente o número de tarefas (*NJobs*), o número de máquinas (*NMachines*), o tipo de problema (*ProblemType*), a característica multi-nível (*MultiLevel*), e o tempo de conclusão óptimo conhecido (*CmaxOptimal*). Estes campos são pesados diferenciadamente na medida de similaridade, definida na equação (16). A medida de similaridade é um valor entre zero (0) e um (1), correspondendo respectivamente a casos nada similares e casos iguais, sendo o resultado de uma soma pesada das similaridades entre os diferentes atributos.

$$\begin{aligned}\mathrm{Sim} = \mathrm{Peso}_{\mathrm{Njobs}} * \mathrm{Sim}_{\mathrm{Njobs}} + \mathrm{Peso}_{\mathrm{Nmachines}} * \mathrm{Sim}_{\mathrm{Nmachines}} + \mathrm{Peso}_{\mathrm{ProbType}} * \mathrm{Sim}_{\mathrm{ProbType}} \\ + \mathrm{Peso}_{\mathrm{MultiLevel}} * \mathrm{Sim}_{\mathrm{MultiLevel}} + \mathrm{Peso}_{\mathrm{CmaxOpt}} * \mathrm{Sim}_{\mathrm{CmaxOpt}}\end{aligned} \quad (16)$$

Na equação (17) estão definidos os pesos de cada atributo.

$$\begin{aligned}&\mathrm{Peso}_{\mathrm{Njobs}} = 0.5;\ \ \mathrm{Peso}_{\mathrm{NMachines}} = 0.25;\ \ \mathrm{Peso}_{\mathrm{ProbType}} = 0.15;\ \mathrm{Peso}_{\mathrm{MultiLevel}} = 0.05; \\ &\mathrm{Peso}_{\mathrm{CmaxOpt}} = 0.05\end{aligned} \quad (17)$$

É dada uma 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 foi considerado um peso de 50% e de 25% para o número de tarefas e número de máquinas respectivamente. Estes dois atributos representam conjuntamente 75% da medida de similaridade. O tipo de problema, com 15%, contém alguma importância para a definição de parâmetros, uma vez que, por exemplo, um problema *Job-Shop* não é igual a um problema de Máquina Única. As características de multi-nível[10] e de tempo de conclusão óptimo não são tão importantes como os restantes atributos, tendo uma importância de 5% cada uma, mas revelam-se importantes para a verificação de casos bastante similares. O atributo de tempo de conclusão óptimo foi considerado para ser possível saber se o novo caso já foi ou não anteriormente processado, pois, quando conhecida, esta é uma característica praticamente única de um problema.

Seguidamente é apresentada a forma de cálculo da similaridade dos diferentes atributos. A similaridade do número de tarefas (equação (18)) e a similaridade do número de máquinas (equação (19)) 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 (20)) e a similaridade da característica multi-nível (equação (21)) podem ser um (1) ou zero (0) caso os atributos sejam iguais ou diferentes, respectivamente. No caso do atributo de tempo de conclusão óptimo, a similaridade deste (equação (22)) é calculada de forma idêntica aos atributos de número de tarefas ou 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.

$$\mathrm{Sim}_{\mathrm{Njobs}} = \frac{\min(\mathrm{Njobs}_1, \mathrm{Njobs}_2)}{\max(\mathrm{Njobs}_1, \mathrm{Njobs}_2)} \quad (18)$$

[10] Quando uma operação é precedida por mais do que uma operação.

$$\mathrm{Sim}_{\mathrm{Nmachines}} = \frac{\min(\mathrm{Nmachines}_1, \mathrm{Nmachines}_2)}{\max(\mathrm{Nmachines}_1, \mathrm{Nmachines}_2)} \tag{19}$$

$$\mathrm{Sim}_{\mathrm{ProbType}} = \begin{cases} 0, & \mathrm{ProbType}_1 \neq \mathrm{ProbType}_2 \\ 1, & \mathrm{ProbType}_1 = \mathrm{ProbType}_2 \end{cases} \tag{20}$$

$$\mathrm{Sim}_{\mathrm{MultiLevel}} = \begin{cases} 0, & \mathrm{MultiLevel}_1 \neq \mathrm{MultiLevel}_2 \\ 1, & \mathrm{MultiLevel}_1 = \mathrm{MultiLevel}_2 \end{cases} \tag{21}$$

$$\mathrm{Sim}_{\mathrm{CmaxOpt}} = \begin{cases} \dfrac{\min(\mathrm{CmaxOpt}_1, \mathrm{CmaxOpt}_2)}{\max(\mathrm{CmaxOpt}_1, \mathrm{CmaxOpt}_2)}, & \mathrm{CmaxOpt}_1 \geq 0 \text{ e } \mathrm{CmaxOpt}_2 \geq 0 \\ 0, & \mathrm{CmaxOpt}_1 < 0 \; ou \; CmaxOpt_2 < 0 \end{cases} \tag{22}$$

5.3.1.3. Fase de Reutilização

Na fase de Reutilização é seleccionado, da lista de casos devolvida pela fase anterior, um caso dos mais similares com o novo caso, sendo a respectiva solução sugerida como uma possível solução para a resolução do mesmo. Assim, a Meta-heurística e os respectivos parâmetros usados para a resolução desse caso muito similar são copiados para a resolução do novo caso.

O algoritmo da fase de Reutilização (Tabela 16) tem a lista de casos devolvida pela fase de Recuperação como parâmetro de entrada e devolve uma Meta-heurística, com respectiva parametrização, como parâmetro de saída. Inicialmente são inicializadas cinco variáveis, nomeadamente uma variável para guardar o melhor caso (*MelhorCaso*), a variável para devolver a Meta-heurística com os parâmetros (*MetaHeuristica*), uma variável para servir de comparação para se determinar se dois casos são muito similares (*SimMuitoSimilares*), uma variável para guardar a lista de melhores casos presentes na *ListaCasos* (*ListaMelhoresCasos*), e uma variável para servir de comparação na determinação dos melhores casos (*RazaoMinima*). Mais à frente é melhor explicado o uso da variável *SimMuitoSimilares* e da variável *RazaoMinima*.

Tabela 16 – Algoritmo da fase de Reutilização

	Entrada: *ListaCasos*
	Saída: *MetaHeuristica* ***// Variável para guardar a Meta-heurística e respectivos parâmetros a retornar***
1	*MelhorCaso ← ({},0);* ***// Variável para guardar o melhor caso***
2	*MetaHeuristica ← {};*
3	*SimMuitoSimilares ← 0.95;* ***// Similaridade para a qual dois casos são considerados muito similares***
4	*ListaMelhoresCasos ←{};* ***// Lista de melhores casos***
5	*RazaoMinima ←0.75;* ***// Razão mínima entre o CmaxOpt e o Cmax, para que um caso possa ser adicionado na ListaMelhoresCasos***

```
6   Se ListaCasos = ∅ Então
7       // Se ListaCasos estiver vazia, aplicar parâmetros pré-definidos pelo utilizador
8       MetaHeuristica = AplicaParametrosPreDefinidos();
9   Senão
10      // Senão, percorrer os casos da ListaCasos
11      Para cada Caso_i ∈ ListaCasos Faz
12          //Para cada caso comparar a respectiva similaridade com a melhor até ao momento
13          Se ComparaSims(Caso_i,MelhorCaso) >= SimMuitoSimilares Então
14              RazaoCaso_i ← Caso_i.CmaxOpt / Caso_i.Cmax;
15              // Perante casos muito similares comparar a razão entre CmaxOpt e Cmax
                com a RazaoMinima
16              Se RazaoCaso_i >= RazaoMinima Então
17                  // Adicionar o Caso_i à ListaMelhoresCasos se a razão entre o seu
                    CmaxOpt e o Cmax for superior ou igual à RazaoMinima
18                  ListaMelhoresCasos.Adicionar(Caso_i);
19              Senão
20                  // Calcular a razão entre o CmaxOpt e o Cmax do MelhorCaso
21                  RazaoMelhor ← MelhorCaso.CmaxOpt / MelhorCaso.Cmax;
22                  // Comparar as duas razões, para se determinar o melhor caso
23                  Se RazaoCaso_i = RazaoMelhor Então
24                      // Razoes iguais, necessário comparar tempos de
                        execução
25                      Se Caso_i.TimeExec< MelhorCaso.TimeExec  Então
26                          // Caso mais eficiente, actualizar MelhorCaso
27                          MelhorCaso ← Caso_i;
28                      Fim
29                  Senão
30                      Se RazaoCaso_i > RazaoMelhor Então
31                          // Caso mais eficaz, actualizar MelhorCaso
32                          MelhorCaso ← Caso_i;
33                      Fim
34                  Fim
35              Fim
36          Senão
37              // Casos não são muito similares, comparar as similaridades directamente
38              Se Caso_i.Sim > MelhorCaso.Sim  Então
39                  // Caso mais similar, actualizar MelhorCaso
40                  MelhorCaso ← Caso_i;
41              Fim
42          Fim
43          // Verificar se a ListaMelhoresCasos tem elementos
44          Se ListaMelhoresCasos <> ∅ Então
45              // Se a ListaMelhoresCasos não estiver vazia, selecciona um caso
                aleatoriamente para ser considerado como melhor caso
```

```
46                          Indice ← aleatório(0,ListaMelhoresCasos.Tamanho) Então
47                          MelhorCaso ← ListaMelhoresCasos.Retorna(Indice);
48                  Fim
49                  // Se a ListaMelhoresCasos estiver vazia, o MelhorCaso é o caso mais similar ou é
                    o caso mais eficaz-eficiente de entre os casos mais similares
50          Fim
51          // Devolver a solução usada pelo MelhorCaso
52          MetaHeuristica ←DevolverSolucao(MelhorCaso);
53  Fim
54  Devolver MetaHeuristica;
```

A primeira verificação a efectuar é se a lista de casos recuperados está ou não vazia. Se estiver vazia, então não existem casos com similaridade superior à superioridade mínima especificada na fase de Recuperação. Isto significa que serão usados os parâmetros pré-definidos pelo utilizador/perito na Interface Gráfica, funcionando como ponto de partida para a resolução de casos futuros com os atributos similares ao novo caso.

Se a lista tiver elementos, então a mesma é percorrida para se seleccionar os casos mais eficazes ou, em contra partida o caso mais similar, se não houver casos suficientemente eficazes. Dentro deste ciclo são efectuadas algumas verificações de modo a determinar se existem casos bastante similares entre si, para, em caso positivo, ser então possível a selecção dos melhores casos ou a selecção do caso mais eficaz-eficiente, através da comparação por tempo de conclusão e tempo de execução. Assim, a verificação dos casos bastante similares é efectuada com recurso à equação (23), 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 bastante similares se a razão entre as suas similaridades com o novo caso for superior a 95%.

$$\mathrm{ComparaSims}(\mathrm{Caso1}, \mathrm{Caso2}) = \frac{\min(\mathrm{Sim}_{\mathrm{Caso1}}, \mathrm{Sim}_{\mathrm{Caso2}})}{\max(\mathrm{Sim}_{\mathrm{Caso1}}, \mathrm{Sim}_{\mathrm{Caso2}})} \tag{23}$$

Quando existem casos bastante similares, é necessário calcular a razão (equação (24)) entre *CmaxOpt* e o *Cmax* de cada caso, para se proceder à sua comparação com a *RazaoMinima*. Esta variável indica que um caso é considerado como um dos melhores se a razão do seu *CmaxOpt* pelo seu *Cmax* for igual ou superior a 0.75. Quando isso acontece, o caso é adicionado na *ListaMelhoresCasos*.

$$\mathrm{RazaoCaso_i} = \frac{\mathrm{CmaxOpt}_{\mathrm{Casoi}}}{\mathrm{Cmax}_{\mathrm{Casoi}}} \tag{24}$$

Quando a razão de um caso não é superior à *RazaoMinima* é necessário calcular a razão entre o *CmaxOpt* e o *Cmax* do melhor caso até dado momento, para ser possível efectuar a comparação na

determinação do melhor caso. No caso da razão do *Caso*$_i$ ser igual à razão do *MelhorCaso* (equação (25)), então os dois casos são igualmente eficazes, sendo necessário comparar os tempos de execução. Se o tempo de execução do *Caso*$_i$ for inferior ao do *MelhorCaso* em dado momento, então o *MelhorCaso* passará a ser o *Caso*$_i$, pois é um caso mais eficiente. Senão, quando a razão do *Caso*$_i$ é superior à razão do MelhorCaso, então o *MelhorCaso* passará a ser o *Caso*$_i$, pois é um caso mais eficaz. Assim, com o valor da razão entre o *CmaxOpt* e o *Cmax* juntamente com o valor do tempo de execução é possível pesar correctamente o binómio eficiência-eficácia.

$$\mathrm{RazaoMelhorCaso} = \frac{\mathrm{CmaxOpt}_{\mathrm{MelhorCaso}}}{\mathrm{Cmax}_{\mathrm{MelhorCaso}}} \tag{25}$$

Quando não existem casos muito similares, a escolha do melhor caso é efectuada através de comparação directa entre as similaridades dos vários casos da lista.

Após a *ListaCasos* ter sido toda percorrida, e se a *ListaMelhoresCasos* não estiver vazia, então é seleccionado 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 seleccionado o melhor caso de entre todos, sempre que um novo caso fosse bastante similar seria continuamente seleccionado o mesmo caso anterior, a não ser que os novos casos obtivessem sempre novos resultados, o que não acontece devido à aleatoriedade subjacente às Meta-heurísticas.

Se a *ListaMelhoresCasos* estiver vazia, o *MelhorCaso* é aquele que se revelou mais similar ou então é o caso mais eficaz-eficiente de entre os casos mais similares.

No final é devolvida a solução na variável *MetaHeuristica*, seja esta a solução do melhor caso ou os parâmetros pré-definidos na Interface Gráfica.

5.3.1.4. Fase de Revisão

Na fase de Revisão, procede-se a uma adaptação da solução sugerida pela fase de Reutilização. A utilização directa das soluções sugeridas leva a que o sistema caia numa estagnação e não consiga evoluir para a obtenção de melhores resultados. Assim, para escapar a soluções óptimas 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. O algoritmo da fase de Revisão está descrito em pseudo-código na Tabela 17.

Tabela 17 – Algoritmo da fase de Revisão

	Entrada: *MetaHeuristica, SimMelhorCaso* **//** **Meta-heurística e similaridade do melhor caso**
	Saída: *MetaHeuristicaRevista* **//** **Meta-heurística com perturbação nos parâmetros**
1	*MetaHeuristicaRevista ← {};*
2	*SimCredMin ← 0.95;* **//** **Variável auxiliar para comparação com a *SimMelhorCaso***
3	*Credito ← 0;* **//** **Variável que guarda o crédito global para ser utilizado na perturbação aos parâmetros**
4	**//** **Inicializar o crédito, de acordo com a comparação da *SimCredMin* com a *SimMelhorCaso***
5	*Se SimCredMin < SimMelhorCaso Então*
6	**//** **Se a *SimMelhorCaso* for superior à *SimCredMin***
7	*Credito←10 +(1-SimCredMin)*100;* **//** **Credito inicializado a 15, corresponde ao crédito mínimo**
8	*Senão*
9	*Credito←10 +(1-SimMelhorCaso)*100;* **//** **Credito inicializado em função da SimMelhorCaso**
10	*Fim*
11	**//** **Calcular os créditos para a inclusão da perturbação em cada parâmetro da Meta-heurística**
12	*ArrayCreditos ← DevolveCreditos(Credito, MetaHeuristica);*
13	**//** **Atribuir a perturbação aos parâmetros da Meta-heurística, em função dos créditos respectivos**
14	*MetaHeuristicaRevista ← ActualizaMetaHeuristica(ArrayCreditos);*
15	*Devolver MetaHeuristicaRevista;*

Este algoritmo recebe dois parâmetros de entrada, nomeadamente a solução sugerida pela fase de Reutilização (isto é, a Meta-heurística sugerida com os respectivos 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, respectivamente, 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 (26)).

$$\text{CreditoMinimo} = 10 + (1 - 0.95) * 100 = 15 \quad (26)$$

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, o 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 efectuado o cálculo desta mesma distribuição, através da chamada ao método *DevolveCreditos(Credito,Metaheuristica)*, descrito na Tabela 18. Com o crédito distribuído para aplicação nos parâmetros, procede-se então à actualização dos parâmetros da Meta-heurística, concluindo-se o algoritmo com a devolução da Meta-heurística actualizada. O algoritmo recebe dois parâmetros, o crédito global e a Meta-heurística com os parâmetros a actualizar. Como parâmetro de saída, devolve um vector com os créditos a serem utilizados em cada um dos parâmetros da Meta-heurística.

Tabela 18 – Algoritmo do método *DevolveCreditos(Credito,MetaHeuristica)* da fase Revisão

```
    Entrada: Credito, MetaHeuristica // Crédito global e a Meta-heurística
    Saída: ArrayCreditos // Vector com os créditos a ser aplicados a cada parâmetro
1   ArrayCreditos ← {};
2   // Percorrer os parâmetros da Meta-heurística
3   Para cada Parametro_i ∈ MetaHeuristica.ListaParametros Faz
4           // Enquanto o crédito global não for nulo
5           Se Credito>0 Então
6                   // Atribuir aleatoriamente crédito ao parâmetro, valor no intervalo [0 ; Credito/2]
7                   ArrayCreditos[Parametro_i] ← aleatório(0, Credito/2);
8                   // Descontar o crédito atribuído ao crédito global
9                   Credito = Credito - ArrayCreditos[Parametro_i];
10                  // Decidir se a perturbação será adicionada ou retirada, com o sinal do crédito
11                  Se aleatório(0,1) = 1 Então
12                          // Se o valor for "verdadeiro" (1) então colocar o crédito com sinal negativo
13                          ArrayCreditos[Parametro_i] = - ArrayCreditos[Parametro_i];
14                  Fim
15          Senão
16                  // Se já não houver crédito, então não é atribuído nenhum ao parâmetro
17                  ArrayCreditos[Parametro_i] = 0;
18          Fim
19  Fim
20  Devolver ArrayCreditos;
```

Assim, depois de inicializado o vector de créditos, procede-se à actualização dos parâmetros, um a um. Logicamente, só se poderá atribuir crédito a um parâmetro se existir crédito global disponível. Assim, essa é a primeira verificação a efectuar. 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 vector e descontado ao crédito global. Seguidamente é feito um cálculo aleatório para ser decidido 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 aumento.

No final, o vector dos créditos dos parâmetros é devolvido.

Nas equações (27) e (28) são apresentadas as fórmulas para actualização dos parâmetros das Meta-heurísticas. A primeira é respeitante à actualização de parâmetros com valores inteiros, enquanto a segunda diz respeito à actualização de parâmetros com vírgula flutuante.

$$\text{Parametro}_i = \text{Parametro}_i + \text{arredondar}\left(\left(\text{Parametro}_i * \frac{\text{ArrayCreditos}[\text{Parametro}_i]}{100}\right), 0\right) \quad (27)$$

$$\text{Parametro}_i = \text{Parametro}_i + \text{arredondar}\left(\left(\text{Parametro}_i * \frac{\text{ArrayCreditos}[\text{Parametro}_i]}{100}\right), 2\right) \quad (28)$$

Assim, nos números inteiros é efectuado um arredondamento, às unidades, da multiplicação do valor do parâmetro com o respectivo 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, com a diferença de que o arredondamento é feito 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.

5.3.1.5. Fase de Retenção

O objectivo da fase de Retenção passa por, tal como a própria designação indica, armazenar o novo caso resolvido na Base de Casos. Esta retenção é constituída por duas fases. Primeiro, são recolhidos os valores dos tempos de conclusão e de execução resultantes da execução do novo caso no Sistema Multi-Agente. De seguida é criado um novo registo na tabela CBR, com todos os dados do caso, sendo também criado um novo registo na tabela da Meta-heurística usada para a resolução do problema, de modo a guardar a solução para uso posterior. Estes dois registos estão relacionados pela sua chave primária.

Com esta fase se conclui 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.

5.3.2. Exemplo Ilustrativo

Nesta secção é apresentado um exemplo que ilustra o funcionamento do sistema de Raciocínio baseado em Casos, do Módulo de Auto-Optimização. O exemplo consiste na chegada de um novo caso ao sistema que, baseado numa Base de Casos, é resolvido através do ciclo de Raciocínio baseado em Casos, com todas as fases explicadas de seguida.

Para este exemplo foram usadas instâncias de problemas de (Fisher & Thompson, 1963) e (Lawrence, 1984).

Novo Caso

O novo caso, definido na Tabela 19, consiste num problema *Job-Shop,* de 10 tarefas e 5 máquinas, e é uma instância do problema *La02* de (Lawrence, 1984).

Tabela 19 – Exemplo Ilustrativo – Novo caso

Número de tarefas (*NJobs*)	Número de máquinas (*NMachines*)	Tipo de problema (*ProblemType*)	Multi-nível (*MultiLevel*)	Tempo de conclusão óptimo (*CmaxOptimal*)
10	5	Job-Shop	não	655

Base de Casos

Na Tabela 20 são apresentados os casos da tabela CBR, sendo possível visualizar todos os campos da mesma, salientando-se a Meta-heurística usada para a execução, bem como a solução inicial e função objectivo. Além dos dados do problema em si, também estão registados os tempos de conclusão e de execução dos casos.

Tabela 20 – Exemplo Ilustrativo – Tabela CBR

ID	MH	NJobs	NMachines	Problem Type	Multi Level	Cmax Optimal	InitSol	ObjFunc	Cmax	Time Exec (s)
1	TS	6	6	JobShop	não	55	SeqNivel	C_{max}	60	0.41
2	SA	10	10	JobShop	não	930	SeqNivel	WT	1385	1.56
3	GA	20	5	JobShop	não	1165	SeqNivel	C_{max}	1720	89.75
4	SA	10	5	JobShop	não	666	SeqNivel	L_{max}	899	2.34
5	GA	10	5	JobShop	não	655	SeqNivel	WT	845	11.85
6	SA	10	5	JobShop	não	597	SeqNivel	WT	793	2.67
7	TS	10	5	JobShop	não	590	SeqNivel	C_{max}	870	46.02
8	TS	6	6	JobShop	não	55	SeqNivel	C_{max}	62	0.83
9	SA	10	10	JobShop	não	930	SeqNivel	WT	1368	1.01
10	GA	20	5	JobShop	não	1165	SeqNivel	C_{max}	1875	95.4
11	SA	10	5	JobShop	não	666	SeqNivel	WT	801	1.76
12	SA	10	5	JobShop	não	655	SeqNivel	WT	838	1.54
13	GA	10	5	JobShop	não	597	SeqNivel	WT	978	12.24
14	SA	10	5	JobShop	não	590	SeqNivel	WT	992	2.03
15	GA	10	5	JobShop	não	666	SeqNivel	WT	795	12.07

Para este exemplo considerou-se uma Base de Casos composta por 15 casos, tendo estes casos sido resolvidos de forma diferenciada.

O preenchimento da Base de Casos foi efectuado da seguinte maneira:

- Os casos do número 1 ao número 7, bem como os do número 8 ao número 14, são instâncias dos problemas *FT06*, *FT10*, *FT20*, *La01*, *La02*, *La03*, e *La04*, respectivamente;
- O caso número 15 é uma instância do problema *La01*;
- Os casos do número 1 ao número 7 foram inicialmente introduzidos com base em resultados de execução já obtidos anteriormente. Os restantes casos foram obtidos através de uma execução normal do sistema, com base nesses casos.

Os parâmetros usados nas Meta-heurísticas são descritos na Tabela 21, Tabela 22, e Tabela 23, e representam, respectivamente, as tabelas TS, GA, e SA.

Tabela 21 – Exemplo Ilustrativo – Tabela TS

ID	StopCrit	NeighGen	SubNeigh	TabuListLen
1	100	0.10	1.00	1
7	100	0.15	1.00	1
8	103	0.15	0.99	1

Tabela 22 – Exemplo Ilustrativo – Tabela GA

ID	NumGen	InitPopGen	PopSize	CrossRate	MutRate
3	100	0.15	1.00	0.75	0.01
5	100	0.10	1.00	0.75	0.01
10	101	0.16	1.00	0.75	0.01
13	105	0.14	1.00	0.75	0.01
15	97	0.16	0.99	0.75	0.01

Tabela 23 – Exemplo Ilustrativo – Tabela SA

ID	StopCrit	NumIteraK	InitTemp	Alpha
2	35	15	15	0.80
4	35	15	15	0.80
6	35	15	15	0.80
9	34	15	15	0.80
11	37	15	15	0.80
12	35	16	15	0.80
14	34	15	15	0.82

Saliente-se que é já possível visualizar alguma perturbação, incluída através da execução normal do sistema, nos casos do número 8 ao número 15. Como existem casos muito similares foi usado um crédito global mínimo (igual a 15), resultando em perturbações mínimas. No entanto são notórias

melhorias nos problemas FT10, La01, e La02, comparando-se os resultados iniciais (casos número 2, 4 e 5) com os resultados após uma execução normal do sistema (casos número 9, 11 e 12).

Fase de Recuperação

A fase de Recuperação, tal como descrita no algoritmo da Tabela 14 (página 115), é composta por duas fases. A primeira fase é uma consulta à Base de Casos para ser efectuada uma pré-selecção de casos. O comando SELECT desta pré-selecção é apresentado na Tabela 24, sendo seleccionados os casos com *Njobs* entre 5 e 20 tarefas, e *Nmachines* entre 1 e 20 máquinas.

Tabela 24 – Exemplo Ilustrativo – Comando SELECT

SELECT c.*
FROM CBR c
WHERE (c.Njobs between 5 and 20) **and** (c.Nmachines between 1 and 20);

Tal como referido na subsecção 5.3.1.2, estes cálculos dos limites inferior e superior são baseados nos pesos da medida de similaridade, e são apresentados nas equações (29), (30), (31), e (32).

$$\text{MinNjobs} = 10 * 0.5 = 5 \tag{29}$$

$$\text{MaxNjobs} = \frac{10}{0.5} = 20 \tag{30}$$

$$\text{MinNmachines} = 5 * 0.25 = 1 \tag{31}$$

$$\text{MaxNmachines} = \frac{5}{0.25} = 20 \tag{32}$$

Com base nos limites definidos no comando de pré-selecção, todos os casos correspondem aos requisitos, sendo todos eles analisados através da segunda fase. Esta segunda fase consiste no cálculo das similaridades dos casos anteriores com o novo caso, sendo adicionados à Lista de Casos aqueles que têm uma similaridade superior à similaridade mínima exigida (0.75).

Na Tabela 25 são apresentadas as similaridades de todos os parâmetros de todos os casos, bem como as similaridades finais resultantes.

As similaridades foram calculadas com base na equação (16), utilizando os pesos definidos na equação (17). As similaridades dos parâmetros foram calculadas, respectivamente, pelas equações (18), (19), (20), (21), e (22).

Tabela 25 – Exemplo Ilustrativo – Casos recuperados da Base de Casos

ID	Sim_{Njobs}	$Sim_{Nmachines}$	$Sim_{ProbType}$	$Sim_{MultiLevel}$	$Sim_{CmaxOpt}$	Similaridade
1	0.3	0.208	0.15	0.05	0.004	0.71
2	0.5	0.125	0.15	0.05	0.035	0.86
3	0.25	0.25	0.15	0.05	0.028	0.72
4	0.5	0.25	0.15	0.05	0.049	0.99
5	0.5	0.25	0.15	0.05	0.05	1.00
6	0.5	0.25	0.15	0.05	0.045	0.99
7	0.5	0.25	0.15	0.05	0.045	0.99
8	0.3	0.208	0.15	0.05	0.004	0.71
9	0.5	0.125	0.15	0.05	0,35	0.86
10	0.25	0.25	0.15	0.05	0.28	0.72
11	0.5	0.25	0.15	0.05	0.49	0.99
12	0.5	0.25	0.15	0.05	0.05	1.00
13	0.5	0.25	0.15	0.05	0.45	0.99
14	0.5	0.25	0.15	0.05	0.45	0.99
15	0.5	0.25	0.15	0.05	0.49	0.99

Como é possível verificar, os casos que contêm uma similaridade superior à similaridade mínima encontram-se destacados, sendo esses inseridos na *ListaCasos*, dando entrada na fase de Reutilização. Também é possível verificar que os casos que têm uma maior similaridade são aqueles que têm 10 tarefas por 5 máquinas, uma vez que estes são os parâmetros que têm mais importância na medida de similaridade.

Fase de Reutilização

Nesta fase são percorridos todos os casos da Lista de Casos resultante da fase anterior, para ser possível seleccionar o melhor caso e devolver a respectiva solução utilizada na sua resolução.

Como é possível verificar pelo algoritmo definido na Tabela 16 da página 118, os melhores casos muito similares são analisados entre si, comparando-se as razões entre os tempos de conclusão óptimos e os tempos de conclusão, tendo por vista seleccionar os melhores casos de entre todos. Estes casos estão apresentados na Tabela 26, com dados relativos à similaridade, tempo de conclusão óptimo, tempo de conclusão e razão entre eles.

Nesta tabela é possível verificar quais são os melhores casos, que correspondem àqueles cuja razão entre *CmaxOpt* e *Cmax* é superior à *RazaoMinima* (0.75), encontrando-se destacados. Tal

como o algoritmo indica, é seleccionado um caso aleatoriamente de entre os melhores casos. A selecção aleatória dos casos destacados na Tabela 26 resultou na indicação do caso 6.

Tabela 26 – Exemplo Ilustrativo – Casos mais similares com o novo caso

ID	Similaridade	CmaxOpt	Cmax	Razão
4	0.99	666	899	0.74
5	1.00	655	845	0.77
6	0.99	597	793	0.75
7	0.99	590	870	0.67
11	0.99	666	801	0.83
12	1.00	655	838	0.78
13	0.99	597	978	0.61
14	0.99	590	992	0.59
15	0.99	666	795	0.83

A solução do melhor caso escolhido aleatoriamente é devolvida pelo algoritmo da fase de Reutilização. Esta solução consiste numa Meta-heurística (neste caso, *Simulated Annealing)* com os respectivos parâmetros, e encontra-se definida na Tabela 27.

Tabela 27 – Exemplo Ilustrativo – Solução do melhor caso

ID	StopCrit	NumIteraK	InitTemp	Alpha
6	35	15	15	0.80

Fase de Revisão

Tal como descrito anteriormente na Tabela 17 (página 122), o algoritmo da fase de Revisão tem como entrada a Meta-heurística sugerida e a similaridade do melhor caso, tendo como objectivo o cálculo de perturbação a ser incluída nos parâmetros.

Uma vez que o melhor caso tem uma similaridade superior a 95%, o crédito global usado para a perturbação foi de 15. Este crédito foi dividido de forma aleatória pelos diferentes parâmetros, tal como descrito no algoritmo *DevolveCreditos(Credito,MetaHeuristica)* da Tabela 18 (página 123).

Na Tabela 28 são apresentados todos os valores usados para o cálculo da perturbação, isto é, os créditos atribuídos a cada parâmetro, as fórmulas de cálculo das actualizações (baseadas nas equações (27) e (28)), bem como o valor final actualizado.

Tabela 28 – Exemplo Ilustrativo – Cálculo da perturbação à solução sugerida

Parâmetro	Crédito atribuído	Cálculo	Valor actualizado
StopCrit	6	35 + 35 * 6 / 100	37
NumIteraK	-5	15 + 15 * (-5) /100	14
InitTemp	3	15 + 15 * 3 /100	15
Alpha	0	0.80 + 0.80 * 0 /100	0.80

Após esta perturbação, o novo caso é executado no sistema, tendo obtido os resultados apresentados na Tabela 29.

Tabela 29 – Exemplo Ilustrativo – Resultados obtidos pela execução do novo caso

C_{max}	TimeExec
830	1.82

Fase de Retenção

Finalmente, e após o caso ter sido executado, é necessário armazená-lo na Base de Casos. Este armazenamento é efectuado na tabela CBR (Tabela 30) e na tabela da Meta-heurística respectiva, neste caso o *Simulated Annealing* (Tabela 31). O novo caso fica com o número 16, uma vez que é de incrementação automática, e encontra-se destacado nas tabelas referidas.

Tabela 30 – Exemplo Ilustrativo – Tabela CBR final

ID	MH	NJobs	NMachines	Problem Type	Multi Level	Cmax Optimal	InitSol	ObjFunc	Cmax	Time Exec (s)
1	TS	6	6	JobShop	não	55	SeqNivel	C_{max}	60	0.41
2	SA	10	10	JobShop	não	930	SeqNivel	WT	1385	1.56
3	GA	20	5	JobShop	não	1165	SeqNivel	C_{max}	1720	89.75
4	SA	10	5	JobShop	não	666	SeqNivel	L_{max}	899	2.34
5	GA	10	5	JobShop	não	655	SeqNivel	WT	845	11.85
6	SA	10	5	JobShop	não	597	SeqNivel	WT	793	2.67
7	TS	10	5	JobShop	não	590	SeqNivel	C_{max}	870	46.02
8	TS	6	6	JobShop	não	55	SeqNivel	C_{max}	62	0.83
9	SA	10	10	JobShop	não	930	SeqNivel	WT	1368	1.01
10	GA	20	5	JobShop	não	1165	SeqNivel	C_{max}	1875	95.4
11	SA	10	5	JobShop	não	666	SeqNivel	WT	801	1.76
12	SA	10	5	JobShop	não	655	SeqNivel	WT	838	1.54
13	GA	10	5	JobShop	não	597	SeqNivel	WT	978	12.24
14	SA	10	5	JobShop	não	590	SeqNivel	WT	992	2.03
15	GA	10	5	JobShop	não	666	SeqNivel	WT	795	12.07
16	SA	10	5	JobShop	não	655	SeqNivel	WT	830	1.82

É possível verificar uma melhoria no tempo de conclusão em relação aos casos anteriores (casos número 5 e 12) da mesma instância (*La02*). Apesar de não ser uma diferença muito significativa, é possível verificar que a parametrização usada levou a uma melhoria da solução.

Tabela 31 – Exemplo Ilustrativo – Tabela SA final

ID	StopCrit	NumIteraK	InitTemp	Alpha
2	35	15	15	0.80
4	35	15	15	0.80
6	35	15	15	0.80
9	34	15	15	0.80
11	37	15	15	0.80
12	35	16	15	0.80
14	34	15	15	0.82
16	37	14	15	0.80

5.4. Sumário

Neste capítulo foi descrito o sistema AutoDynAgents, que consiste num Sistema Multi-Agente concebido para a resolução autónoma, distribuída e cooperativa de problemas de escalonamento sujeitos a perturbações. Foi feita uma descrição de todos os tipos de agentes, assim como os principais módulos presentes na sua arquitectura (secção 5.2.1), nomeadamente o Módulo de Escalonamento (secção 5.2.2) e o Módulo de Adaptação Dinâmica (secção 5.2.3).

Uma vez que se trata do assunto central desta tese, foi dado um especial destaque à descrição do Módulo de Auto-Optimização desenvolvido e integrado do sistema AutoDynAgents, na secção 5.3. Este módulo surgiu como resposta à necessidade da auto-parametrização das Meta-heurísticas presentes no sistema, uma vez que a sua parametrização requer alguma experiência e conhecimento pericial. Para resolução deste problema, foi proposto um sistema de Raciocínio baseado em Casos, e foi descrita a arquitectura deste módulo, com especificação da Base de Casos e de cada fase do ciclo do Raciocínio baseado em Casos. Para um melhor entendimento do funcionamento deste módulo, foi descrito um exemplo ilustrativo na secção 5.3.2.

CAPÍTULO 6. ESTUDO COMPUTACIONAL

6.1. Introdução

Neste capítulo é apresentado e descrito o conjunto de testes ou experiências computacionais com vista à validação do mecanismo de Auto-Optimização proposto. Serão apresentados os resultados computacionais obtidos no âmbito deste trabalho de mestrado, tendo sido usados instâncias de problemas de teste académicos (*Job-Shop*) de Fisher e Thompson (1963) e Lawrence (1984).

A implementação de todo o sistema AutoDynAgents, incluindo o módulo de Auto-Optimização, foi realizada na linguagem de programação Java, tendo sido usada a *framework* Hibernate[11] como camada de acesso à Base de Casos, esta desenvolvida em HSQLDB[12].

A máquina usada para o estudo computacional contém as seguintes características principais: processador Inter Core 2 Quad Q6600 @ 2.40 GHz, 4GB de memória RAM, disco rígido Samsung HD250HJ de 250GB, e sistema operativo Microsoft Windows XP com SP3.

O estudo computacional encontra-se dividido em duas fases principais. Na primeira, foi efectuada uma inicialização da Base de Casos, tendo sido inseridos, para cada instância de problema, os melhores resultados obtidos com o sistema AutoDynAgents até à data, usando as diversas Meta-heurísticas implementadas e para cada uma delas as medidas de optimização (funções objectivo): C_{max} (tempo total de conclusão), WT (atrasos pesados) e L_{max} (atraso total). As Meta-heurísticas utilizadas foram, tal como já referido no capítulo anterior, a Pesquisa Tabu, os Algoritmos Genéticos, o *Simulated Annealing*, *Ant Colony Optimization*, e *Particle Swarm Optimization*, estando as suas parametrizações iniciais definidas para cada problema na secção 6.2.2. Refira-se que para cada execução das Meta-heurísticas, independentemente do critério de optimização utilizado, a solução encontrada é avaliada segundo o critério de avaliação de tempo de conclusão (C_{max}) e tempo de execução (*TimeExec*), tal como apresentado nas tabelas de resultados computacionais. Irá ser usado o termo "combinação Meta-heurística / função objectivo" sempre que se quiser referir a uma Meta-heurística segundo um critério de optimização específico.

Na segunda fase são apresentados os resultados computacionais obtidos. Primeiro encontram-se os resultados que serviram de comparação com os resultados obtidos, estando aqui incluídos os resultados previamente obtidos pelo sistema AutoDynAgents, sem o módulo de Auto-Optimização. Seguidamente são apresentados os resultados obtidos pelo sistema, já com o módulo de Auto-Optimização, sendo tiradas algumas conclusões sobre os mesmos. Por último, são apresentados

[11] https://www.hibernate.org/
[12] http://hsqldb.org/

outros resultados relevantes e respectivas conclusões. A notação usada nas tabelas de resultados está descrita na Tabela 32.

Tabela 32 – Notação das tabelas de resultados computacionais

Coluna	Descrição
Prob.	Instância de problema
n	Número de tarefas
m	Número de máquinas
C_{max} Óptimo	Tempo total de conclusão óptimo
F. Obj.	Critério de optimização, ou função objectivo (C_{max}, WT ou L_{max})
C_{max}	Tempo total de conclusão
WT	Atrasos pesados (*Weighted Tardiness*)
L_{max}	Atraso total
TimeExec (s)	Tempo de execução em segundos

6.2. Inicialização da Base de Casos

Antes de se proceder à extracção de resultados do sistema AutoDynAgents com o módulo de Auto-Optimização desenvolvido no âmbito deste trabalho, foi importante inicializar a Base de Casos com os melhores resultados previamente obtidos pelo sistema. Esta inicialização torna-se importante por ser um ponto de partida para a evolução do módulo de Auto-Optimização.

Os casos inseridos correspondem aos melhores resultados obtidos anteriormente com todas as Meta-heurísticas, para as funções objectivo C_{max}, WT e L_{max}. As parametrizações iniciais das Meta-heurísticas correspondem ao resultado de algum conhecimento pericial e experiências de tentativa-erro.

6.2.1. Casos inseridos

Na Tabela 33 são apresentados os resultados obtidos para problemas de Fisher e Thompson (1963) e de Lawrence (1984). Estes resultados consistem nos melhores tempos de conclusão (C_{max}) e de execução (*TimeExec*, em segundos) obtidos com as Meta-heurísticas Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, Optimização por Colónias de Formigas, e *Particle Swarm Optimization*, para cada instância, com as diferentes funções objectivo (C_{max}, WT e L_{max}) como critério de optimização.

Refira-se que os resultados das Meta-heurísticas Pesquisa Tabu e Algoritmos Genéticos, usados como referência, se encontram publicados em (Madureira et al., 2009b).

Tabela 33 – Tempos de conclusão e tempos de execução dos dados de inicialização da Base de Casos

Prob.	*n*	*m*	C_{max} Óptimo	F. Obj.	Pesquisa Tabu		Algoritmos Genéticos		*Simulated Annealing*		Optimização por Colónia de Formigas		*Particle Swarm Optimization*	
					C_{max}	Time Exec (s)	C_{max}	Time Exec (s)	C_{max}	Time Exec (s)	C_{max}	Time Exec (s)	C_{max}	Time Exec (s)
FT06	6	6	55	C_{max}	59	2.25	65	1.55	60	0.85	59	0.43	69	0.58
				WT	59	2.04	65	1.54	60	0.32	66	0.44	59	0.49
				L_{max}	99	2.43	66	1.54	60	0.4	96	0.38	105	0.5
FT10	10	10	930	C_{max}	1408	12	1448	12.1	1301	0.57	1376	2.9	1379	1.83
				WT	1319	11.96	1427	12.03	1385	0.58	1463	3.07	1319	2.77
				L_{max}	1542	20.39	1399	12.06	1331	0.69	1514	2.8	1306	2.85
FT20	20	5	1165	C_{max}	1707	100.03	1692	89.68	1795	0.77	1509	29.84	1672	8.19
				WT	1664	69.76	1662	91.05	1672	0.62	1454	37.08	1672	8.15
				L_{max}	1686	155.18	1714	89.55	2078	0.6	1457	31.27	1752	9.47
La01	10	5	666	C_{max}	764	12.74	810	12.35	791	0.32	841	1.44	863	2.97
				WT	740	12.24	782	12.3	763	0.32	764	1.57	771	2.85
				L_{max}	947	18.83	888	12.28	792	0.33	1134	1.44	788	3.07
La02	10	5	655	C_{max}	839	13.21	847	11.91	1004	0.31	1004	1.33	876	2.77
				WT	839	13.8	839	11.86	839	0.35	867	1.6	909	2.8
				L_{max}	849	15.94	871	11.88	839	0.32	855	1.3	839	2.86
La03	10	5	597	C_{max}	776	16.61	804	15.33	838	0.33	935	1.57	893	2.87
				WT	749	15.05	809	15.35	793	0.34	954	1.54	847	2.78
				L_{max}	875	21.53	823	15.26	948	0.37	972	1.61	915	2.81
La04	10	5	590	C_{max}	798	12.55	746	12.55	909	0.32	864	1.58	949	2.69
				WT	767	12.5	773	12.54	886	0.31	834	1.6	896	2.73
				L_{max}	842	18.08	822	12.53	876	0.32	937	1.63	851	2.79

Com base nestes resultados, foram inseridos 105 casos na Base de Casos, na tabela CBR, correspondendo à totalidade de execuções efectuadas (7 instâncias de problema x 5 Meta-heurísticas x 3 funções objectivo).

6.2.2. Parametrização Inicial

Seguidamente é descrita a parametrização inicial das Meta-heurísticas para cada instância em análise, nos casos inseridos para a inicialização da Base de Casos.

Para todas as Meta-heurísticas, o algoritmo de geração da solução/indivíduo inicial usado foi o *SeqNivel*, em que cada solução/cromossoma inicial é definida pela ordenação não-decrescente dos

níveis a que pertencem as operações, dando assim prioridade às operações das diferentes tarefas que devem ser processadas primeiro.

Na Pesquisa Tabu e Algoritmos Genéticos foi usada a totalidade da vizinhança/população em todas as instâncias, assim como um critério de paragem/número de gerações de 100. O afastamento máximo e o tamanho da lista tabu foram variáveis de acordo com o problema. Em relação às taxas de cruzamento e de mutação, estas foram definidas como 75% e 1% respectivamente.

Nas restantes Meta-heurísticas não foram implementadas as noções de subvizinhança nem de afastamento máximo, tendo sido utilizado um mecanismo de geração de vizinhança de troca de operações por níveis, que consiste em trocar apenas operações pertencentes ao mesmo nível na gama operatória de modo a evitar a geração de soluções impossíveis.

Para o *Simulated Annealing*, foi utilizada uma temperatura inicial de 15º e um factor de arrefecimento de 80% para todos os problemas. O número de iterações à mesma temperatura e o critério de paragem é variável consoante a instância, uma vez que dependem da dimensão da mesma, de forma proporcional. Na Optimização por Colónia de Formigas varia o número de formigas e o critério de paragem, sendo fixo o número de colónias (1), a taxa de evaporação da feromona (80%) e os critérios *alpha* e *beta* (1). Finalmente, o *Particle Swarm Optimization* só varia no número de partículas e no número de iterações de acordo com a instância, sendo os restantes parâmetros fixos.

A parametrização das Meta-heurísticas encontra-se descrita nas tabelas 34, 35, 36 e 37, sendo respectivamente, para as instâncias FT06, FT10, FT20 e problemas de Lawrence (1984).

Tabela 34 – Parametrização inicial para o problema FT06 (Fisher & Thompson, 1963)

Meta-heurística	Parâmetro	Valor
Pesquisa Tabu	Critério de paragem	100
	Afastamento	10%
	Subvizinhança	100%
	Tamanho da Lista Tabu	1
Algoritmos Genéticos	Número de gerações	100
	Afastamento	10%
	Tamanho da população	100%
	Taxa de cruzamento	75%
	Taxa de mutação	1%
Simulated Annealing	Critério de paragem	25
	Número de iterações à mesma temperatura	10
	Temperatura inicial	15
	Factor de redução da temperatura (*alpha*)	80%
Optimização por Colónia de Formigas	Critério de paragem	50
	Taxa de evaporação	80%

	Número de formigas por colónia	15
	Número de colónias	1
	Alpha	1
	Beta	1
Particle Swarm Optimization	Número de iterações	500
	Número de partículas	15
	Velocidade mínima	-4
	Velocidade máxima	4
	Inércia mínima	40%
	Inércia máxima	95%
	C1	2.0
	C2	2.0
	Limite inferior	0
	Limite superior	4

Tabela 35 – Parametrização inicial para o problema FT10 (Fisher & Thompson, 1963)

Meta-heurística	Parâmetro	Valor
Pesquisa Tabu	Critério de paragem	100
	Afastamento	15%
	Subvizinhança	100%
	Tamanho da Lista Tabu	1
Algoritmos Genéticos	Número de gerações	100
	Afastamento	15%
	Tamanho da população	100%
	Taxa de cruzamento	75%
	Taxa de mutação	1%
Simulated Annealing	Critério de paragem	35
	Número de iterações à mesma temperatura	15
	Temperatura inicial	15
	Factor de redução da temperatura (*alpha*)	80%
Optimização por Colónia de Formigas	Critério de paragem	100
	Taxa de evaporação	80%
	Número de formigas por colónia	25
	Número de colónias	1
	Alpha	1
	Beta	1
Particle Swarm Optimization	Número de iterações	1000
	Número de partículas	25
	Velocidade mínima	-4
	Velocidade máxima	4
	Inércia mínima	40%
	Inércia máxima	95%

	C1	2.0
	C2	2.0
	Limite inferior	0
	Limite superior	4

Tabela 36 – Parametrização inicial para o problema FT20 (Fisher & Thompson, 1963)

Meta-heurística	Parâmetro	Valor
Pesquisa Tabu	Critério de paragem	100
	Afastamento	15%
	Subvizinhança	100%
	Tamanho da Lista Tabu	3
Algoritmos Genéticos	Número de gerações	100
	Afastamento	15%
	Tamanho da população	100%
	Taxa de cruzamento	75%
	Taxa de mutação	1%
Simulated Annealing	Critério de paragem	50
	Número de iterações à mesma temperatura	15
	Temperatura inicial	15
	Factor de redução da temperatura (*alpha*)	80%
Optimização por Colónia de Formigas	Critério de paragem	250
	Taxa de evaporação	80%
	Número de formigas por colónia	50
	Número de colónias	1
	Alpha	1
	Beta	1
Particle Swarm Optimization	Número de iterações	2000
	Número de partículas	35
	Velocidade mínima	-4
	Velocidade máxima	4
	Inércia mínima	40%
	Inércia máxima	95%
	C1	2.0
	C2	2.0
	Limite inferior	0
	Limite superior	4

Tabela 37 – Parametrização inicial para os problemas de Lawrence (1984)

Meta-heurística	Parâmetro	Valor
Pesquisa Tabu	Critério de paragem	100
	Afastamento	15%
	Subvizinhança	100%

	Tamanho da Lista Tabu	1
Algoritmos Genéticos	Número de gerações	100
	Afastamento	10%
	Tamanho da população	100%
	Taxa de cruzamento	75%
	Taxa de mutação	1%
Simulated Annealing	Critério de paragem	35
	Número de iterações à mesma temperatura	15
	Temperatura inicial	15
	Factor de redução da temperatura (*alpha*)	80%
Optimização por Colónia de Formigas	Critério de paragem	100
	Taxa de evaporação	80%
	Número de formigas por colónia	25
	Número de colónias	1
	Alpha	1
	Beta	1
Particle Swarm Optimization	Número de iterações	1000
	Número de partículas	25
	Velocidade mínima	-4
	Velocidade máxima	4
	Inércia mínima	40%
	Inércia máxima	95%
	C1	2.0
	C2	2.0
	Limite inferior	0
	Limite superior	4

Com base nestas parametrizações para os vários problemas, foram inseridas as parametrizações para os 105 casos inseridos na Base de Casos, nas respectivas tabelas (TS, GA, SA, ACO, e PSO), descritas na subsecção 5.3.1.1.

6.3. Resultados Computacionais

Nesta secção são apresentados os resultados computacionais obtidos. Inicialmente apresentam-se os resultados que servem de base para a comparação, seguidamente os resultados obtidos, e finalmente outros resultados considerados relevantes.

Tabela 38 – Resultados comparativos de MAPS-TCS, MAPS-LCS, Pesquisa Tabu, Algoritmos Genéticos, *Simulated Annealing*, Optimização por Colónia de Formigas, e *Particle Swarm Optimization*

Prob.	*n*	*m*	C_{max} Óptimo	MAPS – TCS			MAPS – LCS			F. Obj.	Pesquisa Tabu			Algoritmos Genéticos			*Simulated Annealing*			Optimização por Colónia de Formigas			*Particle Swarm Optimization*		
				Médio	Melhor	Pior	Médio	Melhor	Pior		Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior
FT06	6	6	55	69.6	62	82	65.8	57	76	C_{max}	61.4	59	64	78.4	65	97	67.2	60	71	69.0	59	83	86.4	69	93
										WT	63.2	59	68	74.0	65	89	62.4	60	65	75.6	66	80	63.2	59	66
										L_{max}	99.0	99	99	74.6	66	96	60.0	60	60	105.2	96	116	141.0	105	177
FT10	10	10	930	1203.4	1164	1242	1277	1172	1373	C_{max}	1519.0	1408	1612	1476.4	1448	1540	1385.0	1301	1426	1395.0	1376	1432	1405.6	1379	1451
										WT	1368.8	1319	1416	1627.2	1427	1816	1372.8	1385	1398	1546.6	1463	1590	1328.8	1319	1347
										L_{max}	1634.8	1542	1739	1504.6	1399	1611	1331	1331	1331	1527.0	1514	1531	1380.0	1306	1462
FT20	20	5	1165	1603.4	1513	1723	1470	1417	1503	C_{max}	1715.0	1707	1726	1761.4	1692	1952	2011.2	1795	2091	1615.4	1509	1706	1713.2	1672	1733
										WT	1678.8	1664	1714	1696.0	1662	1755	1837.8	1672	2057	1639.6	1454	1782	1879.2	1672	2117
										L_{max}	1784.6	1686	1889	1740.6	1714	1833	2078.0	2078	2078	1517.4	1457	1602	1921.2	1752	2074
La01	10	5	666	782	725	832	861	822	891	C_{max}	812.2	764	858	919.8	810	1160	854.6	791	883	923.6	841	1005	876.4	863	888
										WT	772.6	740	795	865.0	782	921	876.2	763	976	833.0	764	910	935.0	771	976
										L_{max}	1059.0	947	1122	1049.2	888	1164	821.4	792	841	1184.5	1134	1235	858.0	788	887
La02	10	5	655	886.6	840	911	821.6	744	873	C_{max}	869.6	839	920	914.4	847	985	1063.0	1004	1128	1068.0	1004	1132	902.4	876	935
										WT	859.4	839	876	940.8	839	1117	911.0	839	929	1055.5	867	1270	975.4	909	1034
										L_{max}	968.4	849	1099	979.0	871	1152	839	839	839	985.0	855	1063	880.0	839	900
La03	10	5	597	778.8	726	823	793.6	716	858	C_{max}	834.6	776	916	888.0	804	997	875.2	838	888	1172.4	935	1549	924.0	893	984
										WT	794.2	749	830	849.0	809	894	898.2	793	942	1076.8	954	1143	877.2	847	932
										L_{max}	947.4	875	990	898.2	823	988	970.8	948	986	1940.0	972	2158	960.8	915	1001
La04	10	5	590	835	801	894	786.2	724	819	C_{max}	865.8	798	924	904.8	746	979	977.6	909	1025	1046.0	864	1228	988.6	949	1087
										WT	875.2	767	989	846.6	773	950	971.6	886	993	973.0	834	1112	938.4	896	949
										L_{max}	959.2	842	1085	915.4	822	1003	876	876	876	1793.4	937	2632	881.8	851	928

6.3.1. Resultados prévios

Na Tabela 38 são apresentados os resultados prévios obtidos antes da implementação do módulo de Auto-Optimização, que servem de comparação aos resultados obtidos com o sistema AutoDynAgents com o módulo de Auto-Optimização incluído.

Apresentam-se também os resultados do sistema MAPS (Wellner & Dilger, 1999) na resolução das mesmas instâncias, para ser possível comparar os resultados obtidos com outro sistema referido na literatura. O sistema MAPS apresenta duas abordagens: *Tightly Coupled Scheduling* (TCS) e *Loosely Coupled Scheduling* (LCS).

Todos os resultados foram obtidos após cinco execuções para cada instância considerada, usando cada um dos critérios de optimização referidos, sendo apresentados o valor médio, a melhor e a pior solução obtida.

Para o **problema FT06**, é notória uma grande proximidade da qualidade das soluções encontradas pelas diferentes Meta-heurísticas, embora algumas tenham obtido melhores soluções do que outras, mas não evidenciando uma diferença considerável, excepto em alguns casos quando usando a função objectivo L_{max} como critério de optimização (em Pesquisa Tabu, Optimização por Colónia de Formigas e *Particle Swarm Optimization*). É importante também salientar que, em termos de resultados médios, a Pesquisa Tabu (com C_{max} e WT), o *Simulated Annealing* (com WT e L_{max}), e o *Particle Swarm Optimization* (com WT) obtiveram melhor desempenho do que o sistema MAPS.

No **problema FT10** já é possível observar diferenças um pouco consideráveis entre o desempenho das Meta-heurísticas, quando usando critérios de optimização (funções objectivo) diferentes. A função objectivo L_{max}, quando usada nas Meta-heurísticas Pesquisa Tabu e Optimização por Colónia de Formigas, obteve piores resultados do que os outros critérios de optimização, na maioria das vezes. O melhor resultado foi obtido pela combinação *Simulated Annealing* com função objectivo $C_{max,}$ mas mesmo assim um pouco distante do sistema MAPS.

Aumentando o número de tarefas, para o **problema FT20**, nota-se um maior equilíbrio de resultados, excepto no desempenho do *Simulated Annealing* na optimização do atraso máximo (L_{max}). É de salientar que, após a observação de três instâncias diferentes (6, 10 e 20 tarefas), os piores resultados médios obtidos tiveram sempre associado o critério de optimização L_{max}, sendo um indicador de que esta será, talvez, a função objectivo menos adequada. Ainda no problema FT20, a Meta-heurística Optimização por Colónia de Formigas revelou uma eficácia muito elevada, revelando-se assim adequada para problemas de maiores dimensões.

Passando para a classe de **problemas de Lawrence**, com 10 tarefas e 5 máquinas, é possível observar que a maioria das Meta-heurísticas revelou uma eficácia significativa na minimização dos atrasos pesados e do atraso máximo. Nota-se também que as Meta-heurísticas Pesquisa Tabu,

Algoritmos Genéticos, e Optimização por Colónia de Formigas obtiveram os piores resultados na minimização de L_{max}. Existindo um equilíbrio bastante elevado no comportamento das Meta-heurísticas, é no entanto possível notar vantagens no uso de Pesquisa Tabu e *Simulated* Annealing nas instâncias La01, La02 e La03, e no uso de Algoritmos Genéticos e Pesquisa Tabu na instância La04. Fazendo uma comparação com o sistema MAPS, é possível observar que algumas combinações Meta-heurística / função objectivo obtiveram melhores resultados do que uma abordagem do MAPS, e, embora a outra tenha sido melhor, ficaram também bastante próximas. Quando analisados os resultados médios, a Pesquisa Tabu obteve melhor desempenho do que o sistema MAPS, revelando uma maior consistência. Por exemplo, na instância La01, todas as Meta-heurísticas na minimização de WT obtiveram melhores resultados do que a abordagem LCS, e, embora a abordagem TCS tenha obtido um melhor resultado, quando comparando resultados médios, nota-se uma vantagem da Pesquisa Tabu na minimização da função objectivo WT.

Perante esta análise, é possível retirar as seguintes conclusões:

- Existe um grande equilibrio no desempenho das diferentes Meta-heurísticas em análise, na resolução de problemas de pequenas dimensões;
- Em problemas de grandes dimensões, denota-se a vantagem revelada pela Meta-heurística Optimização por Colónia de Formigas;
- Nos problemas de média dimensão, a Pesquisa Tabu revelou grande eficácia, sendo seguida pelo *Simulated Annealing*, sendo que se revelam boas Meta-heurísticas para esta classe de problemas;
- De salientar a grande consistência no comportamento das Meta-heurísticas, em particular da Pesquisa Tabu, que obteve melhores resultados médios do que o sistema MAPS, no problema La01.

De notar que, com uma simples implementação das Meta-heurísticas e com um pequeno esforço de parametrização, foi possível obter um bom desempenho na maioria das instâncias de problemas, quando comparando com o sistema MAPS.

6.3.2. Resultados obtidos e conclusões

Seguidamente são apresentados os resultados obtidos pelo sistema AutoDynAgents, após a inclusão do módulo de Auto-Optimização, bem como retiradas algumas conclusões sobre os mesmos.

Para todas as instâncias foram executadas 100 corridas e são apresentados resultados após 25, 50, 75 e 100 execuções, na Tabela 39 e na Tabela 40, para cada uma delas.

Tabela 39 – Resultados computacionais dos vários problemas com Auto-Optimização após 25 e 50 execuções

Prob.	*n*	*m*	C_{max} Óptimo	Após 25 execuções						Após 50 execuções					
				C_{max}			TimeExec (s)			C_{max}			TimeExec (s)		
				Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior
FT06	6	6	55	60.33	59	68	0.26	0.22	0.5	60.4	59	68	0.24	0.2	0.5
FT10	10	10	930	1385.65	1276	1485	25.71	23.57	27.45	1384.62	1276	1485	24.71	23.04	27.45
FT20	20	5	1165	1675.76	1641	1741	4.42	4.1	5.15	1679.43	1641	1741	4.23	3.79	5.15
La01	10	5	666	797.25	747	862	13.44	11.05	15.94	797.99	747	908	12.97	10.72	15.94
La02	10	5	655	880.21	825	995	17.56	15.6	18.5	877.06	808	995	17.16	15.6	18.5
La03	10	5	597	930.09	818	1020	71.66	64.7	73.61	939.63	818	1020	70.72	64.7	73.61
La04	10	5	590	917	802	992	15.68	14.58	16.65	909.64	760	992	15.03	13.1	16.65

Tabela 40 – Resultados computacionais dos vários problemas com Auto-Optimização após 75 e 100 execuções

Prob.	*n*	*m*	C_{max} Óptimo	Após 75 execuções						Após 100 execuções					
				C_{max}			TimeExec (s)			C_{max}			TimeExec (s)		
				Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior
FT06	6	6	55	60.12	59	68	0.24	0.2	0.5	60.11	59	68	0.23	0.2	0.5
FT10	10	10	930	1390.16	1276	1485	25.22	23.04	27.45	1383.7	1276	1485	25.47	23.04	27.45
FT20	20	5	1165	1679.32	1628	1741	4.02	3.27	5.15	1679.12	1612	1741	4.11	3.27	5.15
La01	10	5	666	793.42	747	908	12.76	10.41	15.94	787.97	740	908	12.59	10.13	15.94
La02	10	5	655	869.35	808	995	16.68	15.11	18.5	867.6	808	995	16.37	14.9	18.5
La03	10	5	597	938.59	747	1020	70.07	64.7	73.61	932.52	747	1020	70.79	64.7	75.32
La04	10	5	590	913.09	760	992	14.41	12.18	16.65	910.43	760	1001	13.96	11.96	16.65

Seguidamente é realizada uma análise dos resultados obtidos, para cada instância, primeiro apenas avaliando a evolução ao longo do tempo, e depois comparando os resultados finais com os resultados previamente obtidos.

Começando por analisar a **instância FT06**, nota-se que os resultados melhor e pior foram obtidos nas primeiras 25 execuções. Na generalidade os resultados são bastante bons e com uma consistência elevada, como se pode analisar pelos resultados médios. Nesse aspecto há uma degradação ao fim de 50 execuções, mas existem também melhorias ao fim de 75 e 100 execuções, ficando a grande maioria dos resultados à volta do C_{max} = 60. Relativamente a tempos de execução, nota-se também grande consistência, ficando a maioria à volta dos 0,25 segundos.

Passando para o **problema FT10**, e em semelhança ao problema FT06, obtiveram-se ambos os resultados melhor e pior nas primeiras 25 execuções. Em relação a resultados médios, houve uma melhoria muito ligeira das 25 para as 50 execuções, notando-se uma degradação já algo significativa após as 75 execuções. No entanto, nas últimas 25 execuções, houve uma melhoria significativa de resultados, como se pode avaliar pelo resultado médio obtido ao final das 100 execuções. Nota-se que a resolução desta instância já demora mais tempo, situando-se à volta dos 25 segundos, por ser um problema de maior dimensão comparativamente ao FT06.

Na **instância FT20**, por ser um problema de grandes dimensões, começam-se a evidenciar os pequenos tempos de execução comparativamente ao problema FT10, estando estes à volta dos 4 segundos. Pode parecer estranho, pois logicamente um problema de maiores dimensões demoraria mais tempo a ser resolvido do que um problema mais pequeno, mas na realidade, e uma vez que o sistema AutoDynAgents usa várias Meta-heurísticas, é usada outra Meta-heurística para a resolução deste problema, como será explicado mais à frente. Em relação aos tempos de conclusão, e contrariamente ao que se tinha verificado nas duas instâncias anteriores, o melhor resultado só foi obtido ao fim de 100 execuções, salientando-se assim o facto de o sistema ser capaz de evoluir para melhores resultados. Ao nível de resultados médios, nota-se uma grande consistência, embora tenha havido uma ligeira degradação ao fim das 50, 75 e 100 execuções, comparativamente às primeiras 25 execuções.

Passando para os problemas de Lawrence (1984), na **instância La01** não se notam melhorias nos resultados médios ao fim de 50 execuções, mas os mesmos já melhoram no final de 75 execuções, continuando essa melhoria nas últimas 25 execuções. Nota-se também que, além dos resultados médios, também se melhorou o melhor resultado, nas últimas 25 execuções. O pior resultado piora após 50 execuções e mantém-se até ao final das 100 execuções. Considerando-se os tempos de execução, nota-se uma diferença de alguns segundos entre os melhores e os piores resultados, devido ao uso de diferentes técnicas, mas evidencia-se também uma ligeira melhoria dos tempos de execução médios ao longo do tempo, facto que ocorre possivelmente pelo uso de Meta-heurísticas mais rápidas, com critérios de paragem mais reduzidos, comparativamente às parametrizações iniciais. Isto é também válido para os restantes problemas de Lawrence (1984).

Na **instância La02**, é notória uma ligeira melhoria na obtenção do melhor resultado, ao fim de 50 execuções. Em relação ao pior resultado, este foi obtido durante as primeiras 25 execuções, não se verificando uma deterioração do mesmo. Ao nível dos resultados médios, notam-se melhorias no final de cada grupo de execuções, sendo que no final de todas as execuções já se verificam melhorias algo significativas, relativamente às primeiras 25 execuções.

Para a **instância La03**, e contrariamente ao que se verificou anteriormente, houve uma deterioração dos resultados médios ao longo do tempo, quando comparado com as primeiras 25 execuções. No entanto, no final das 100 execuções, notam-se melhorias significativas em relação às execuções anteriores (50 e 75). O pior resultado mantém-se inalterado até ao final das execuções, mas o melhor resultado melhora algo significativamente no final das 75 execuções. Nota-se que, na resolução desta instância, houve tempos de execução superiores a 70 segundos, tendo sido a instância que mais tempo demorou a ser resolvida, facto que aconteceu devido às características do próprio problema e também às técnicas e parametrização utilizada.

Finalmente, na **instância La04**, é notória uma melhoria dos resultados médios ao fim de 50 execuções, havendo uma deterioração ao fim de 75 execuções, voltando a notar-se uma melhoria no final das 100 execuções. O melhor resultado foi obtido no final das 50 execuções, notando-se uma melhoria em relação às primeiras 25 execuções. Contrariamente aos problemas anteriores, o pior resultado foi obtido nas últimas 25 execuções.

Após a análise de cada um dos problemas, é possível chegar a algumas **conclusões**:

- O sistema AutoDynAgents com o módulo de Auto-Optimização é capaz de evoluir na obtenção de melhores resultados, ao longo do tempo, como se verificou nas instâncias FT20, La01, La02, La03, e La04;
- Não se verifica deterioração nos piores resultados ao longo do tempo, na maioria dos problemas, com excepção das instâncias La01 e La04;
- Nota-se uma grande consistência nos resultados médios, havendo melhorias significativas ao longo do tempo, na maioria das instâncias, com excepção de FT20 e La03;
- A nível de tempos de execução, nota-se também alguma consistência ao longo do tempo, evidenciando-se inclusive que os tempos de execução médios baixaram ao longo do tempo, devido ao facto do módulo escolher casos de acordo com a sua eficiência-eficácia.

Seguidamente irá ser feita uma análise aos resultados obtidos comparativamente com os resultados prévios, da Tabela 38. Realizar-se-á primeiro uma comparação com os resultados do sistema AutoDynAgents sem módulo de Auto-Optimização, sendo de seguida comparados os resultados obtidos com o sistema MAPS, para cada problema.

Assim, para a **instância FT06**, foi obtido igual valor para os melhores resultados (Figura 37), tendo-se verificado uma maior consistência em relação aos resultados médios (Figura 36). O pior resultado (Figura 38) foi melhor do que algumas combinações Meta-heurística / função objectivo. Comparando com o sistema MAPS, verifica-se uma clara vantagem em relação à abordagem TCS, tendo sido obtidas melhorias tanto no resultado médio como no melhor e pior resultado. Relativamente à abordagem LCS, não se conseguiu obter um melhor resultado, mas melhorou-se o resultado médio e o pior resultado.

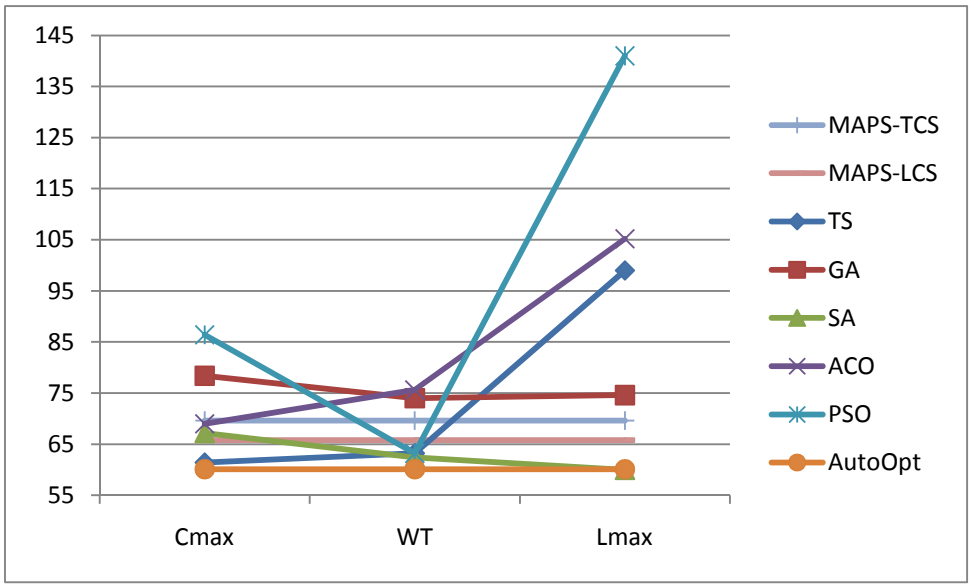

Figura 36 – Gráfico comparativo dos resultados médios obtidos para o problema FT06

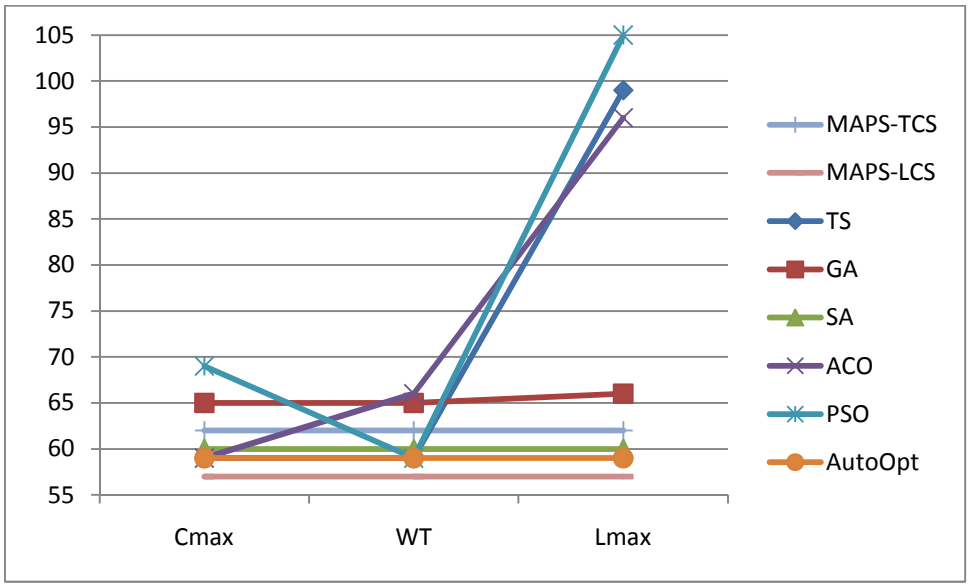

Figura 37 – Gráfico comparativo dos melhores resultados obtidos para o problema FT06

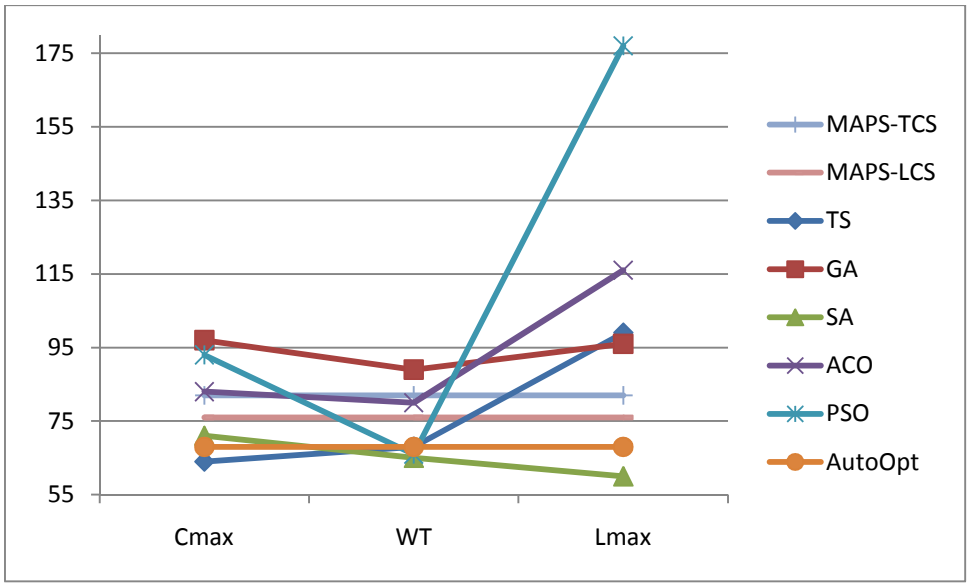

Figura 38 – Gráfico comparativo dos piores resultados obtidos para o problema FT06

Passando para **FT10**, conseguiu-se obter um melhor resultado (Figura 40) que nunca tinha sido obtido anteriormente, isto é, o valor 1276 em oposição ao valor 1301 da combinação *Simulated Annealing* / C_{max}, notando-se aqui já vantagens do uso do módulo de Auto-Optimização. Ao nível do pior resultado (Figura 41) não se notou melhorias em relação à maioria dos resultados prévios. Em relação aos resultados médios (Figura 39), estes são melhores em relação a algumas combinações, mas também ligeiramente piores em relação a outras, pelo que os resultados não são conclusivos. Comparando com o sistema MAPS, nota-se que este obteve claramente melhores resultados, não se conseguindo ainda sequer aproximar dos resultados médios, em nenhuma das abordagens.

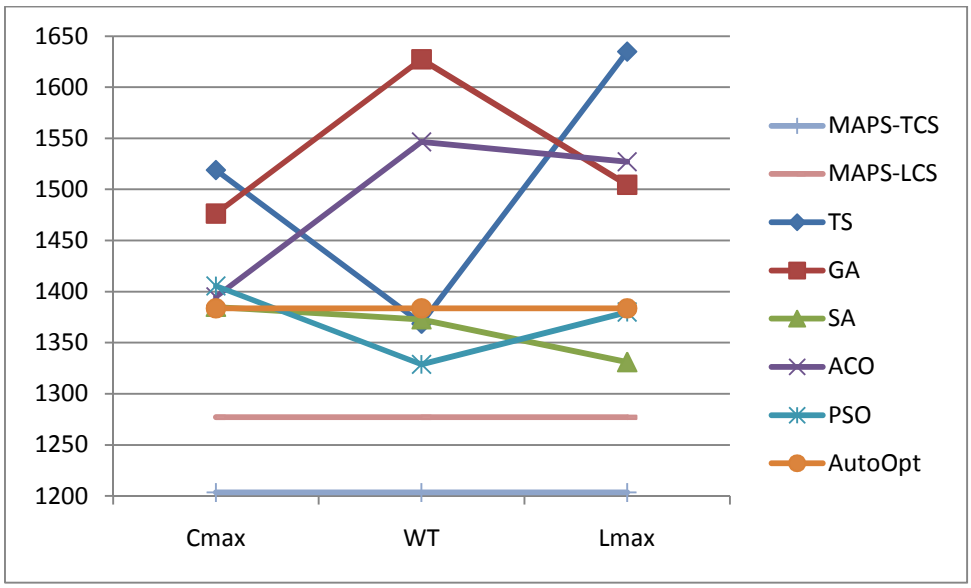

Figura 39 – Gráfico comparativo dos resultados médios obtidos para o problema FT10

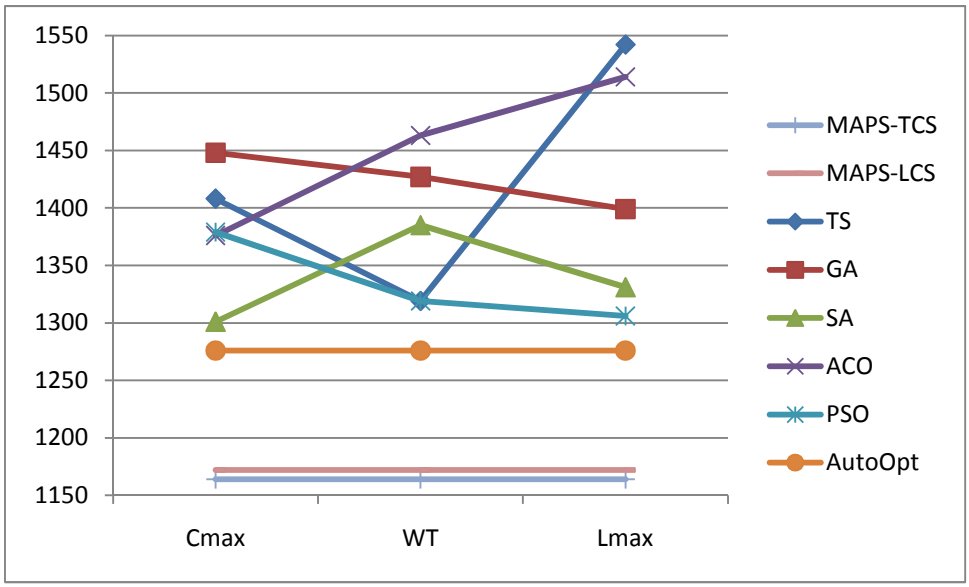

Figura 40 – Gráfico comparativo dos melhores resultados obtidos para o problema FT10

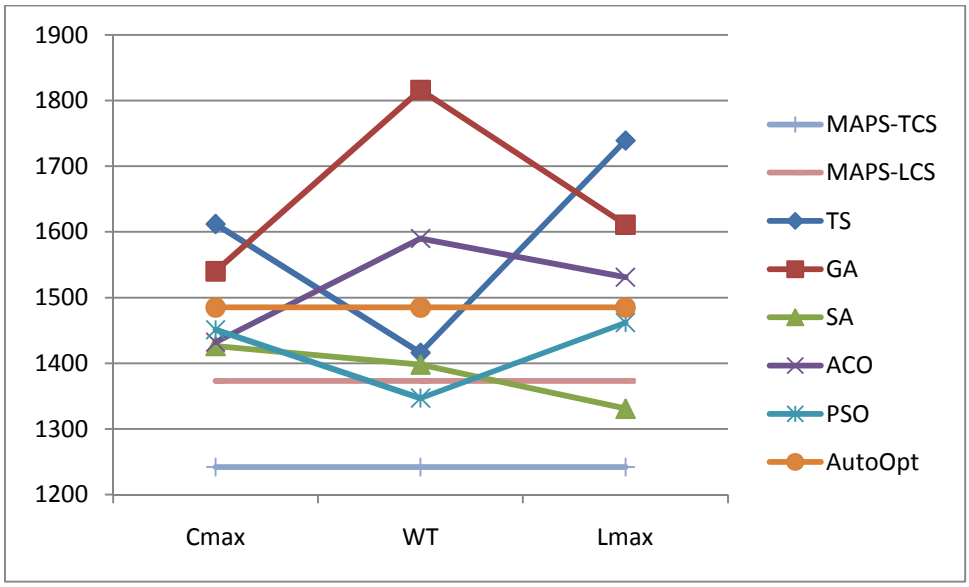

Figura 41 – Gráfico comparativo dos piores resultados obtidos para o problema FT10

Para o problema de maiores dimensões, a **instância FT20** (20 tarefas e 5 máquinas), foi obtido um melhor resultado em relação à maioria das combinações (Figura 43). Apenas não se conseguiu melhores resultados comparando com os anteriores resultados da Optimização por Colónia de Formigas. Para os piores resultados (Figura 44), nota-se alguma melhoria comparativamente à maioria das combinações. Relativamente aos resultados médios (Figura 42), e em semelhança aos

melhores resultados, apenas a Optimização por Colónia de Formigas obteve resultados melhores. Assim, neste problema, não se notou uma melhoria significativa no uso do módulo de Auto-Optimização, uma vez que a Optimização por Colónia de Formigas, com a parametrização inicial, obteve resultados bastante melhores. Comparativamente ao sistema MAPS, houve uma aproximação em relação à abordagem TCS, embora ainda se tenha ficado distante da outra abordagem.

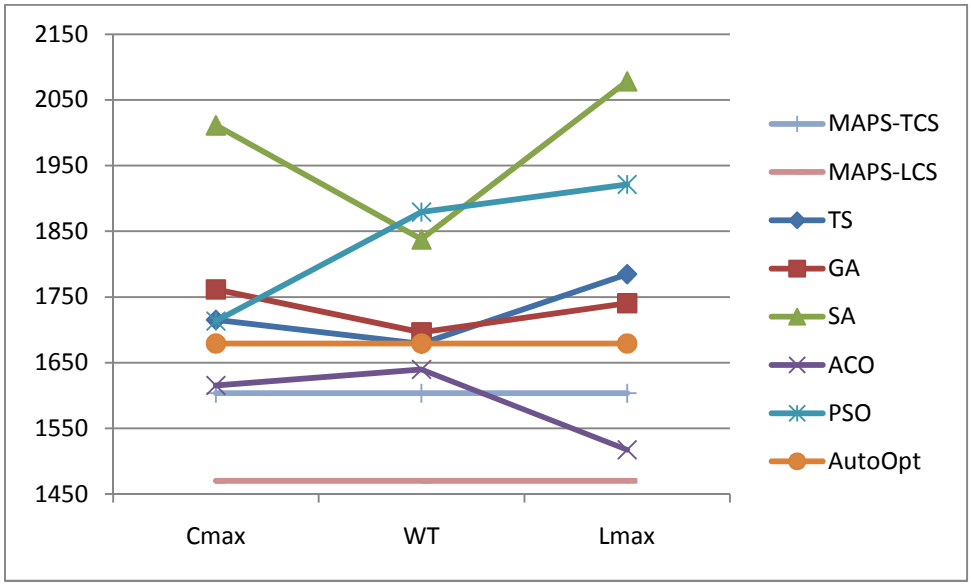

Figura 42 – Gráfico comparativo dos resultados médios obtidos para o problema FT20

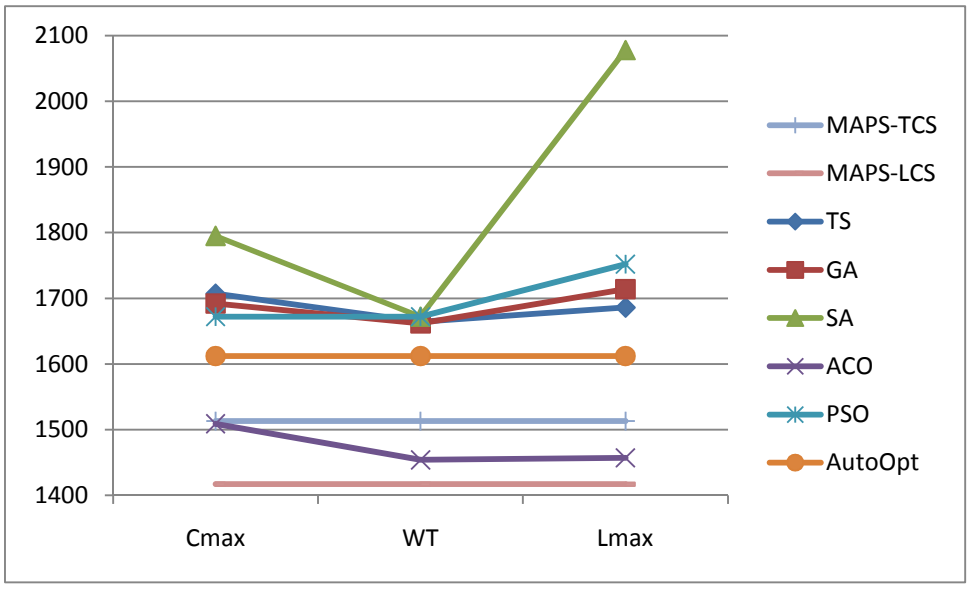

Figura 43 – Gráfico comparativo dos melhores resultados obtidos para o problema FT20

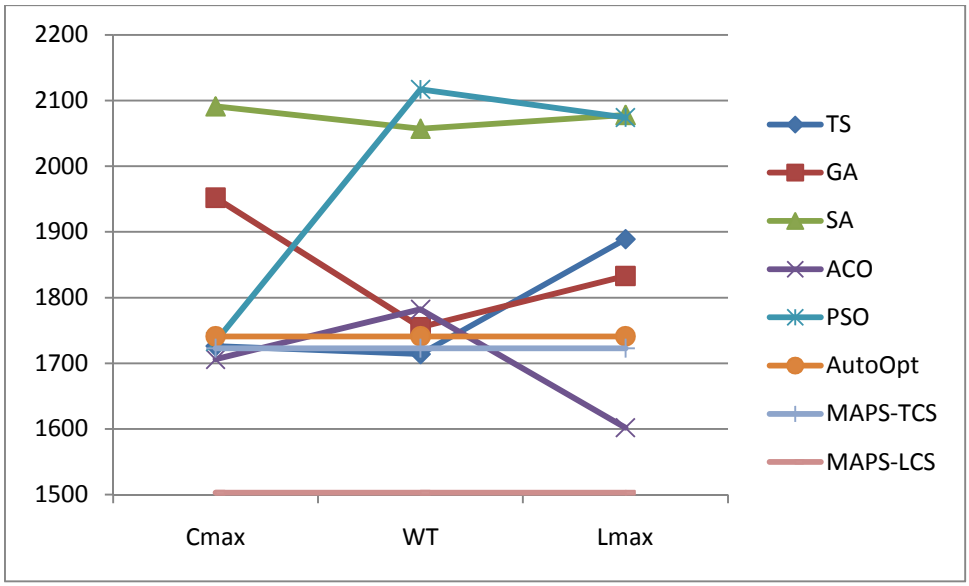

Figura 44 – Gráfico comparativo dos piores resultados obtidos para o problema FT20

Passando para os problemas de Lawrence (1984), na **instância La01** conseguiu obter-se um melhor resultado igual (Figura 46), com um valor de 740, que tinha sido obtido por Pesquisa Tabu com minimização de WT. O pior resultado obtido foi melhor do que a maioria das combinações (Figura 47). Em relação aos resultados médios (Figura 45), estes foram melhores do que a grande maioria das combinações, apenas inferiores à combinação Pesquisa Tabu / WT. Comparando com o sistema MAPS, observa-se melhores resultados em relação à abordagem LCS, embora sejam piores resultados comparando com a abordagem TCS.

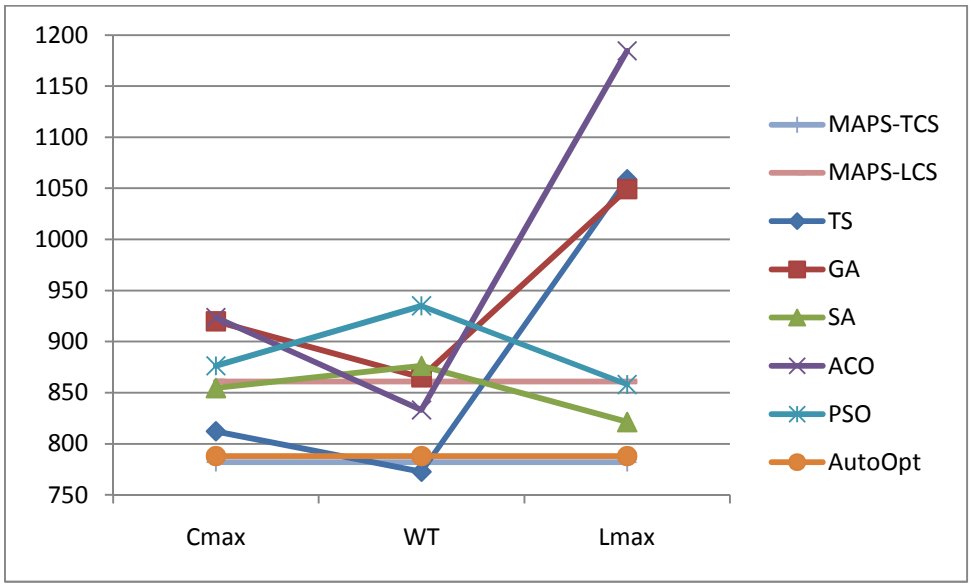

Figura 45 – Gráfico comparativo dos resultados médios obtidos para o problema La01

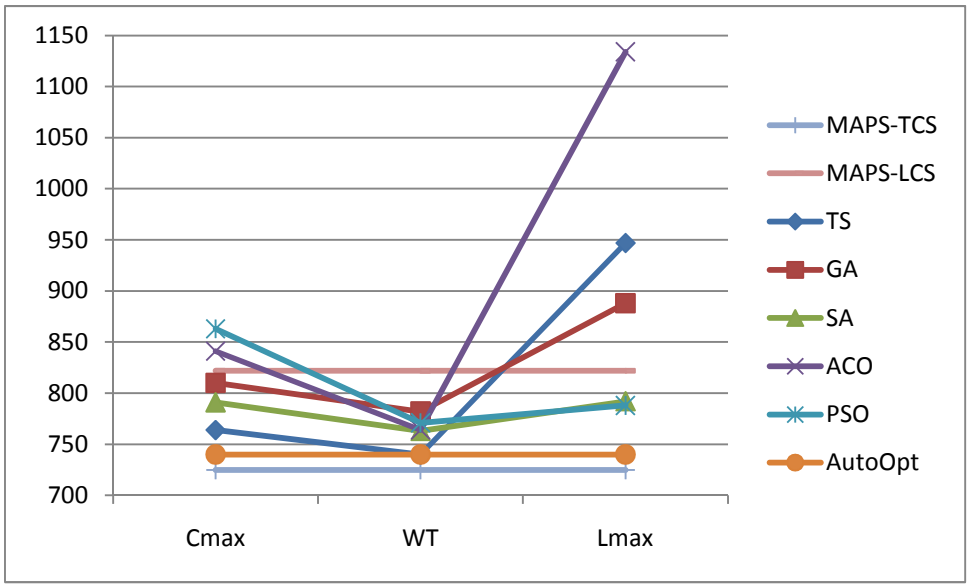

Figura 46 – Gráfico comparativo dos melhores resultados obtidos para o problema La01

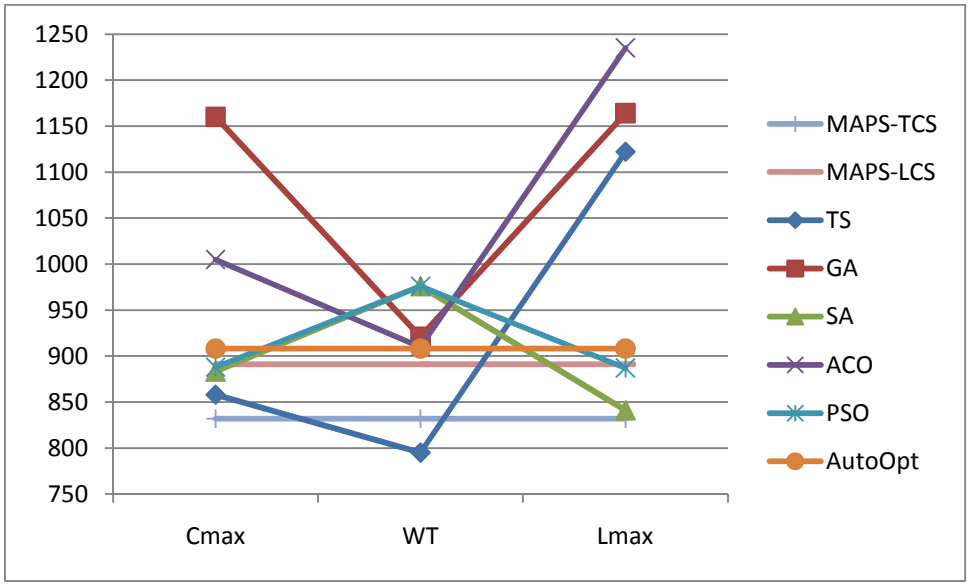

Figura 47 – Gráfico comparativo dos piores resultados obtidos para o problema La01

Na **instância La02**, e em semelhança ao problema FT10, conseguiu-se obter um melhor resultado que nunca antes tinha sido atingido, isto é, o valor 808 em oposição ao valor 839 obtido por grande parte das combinações (Figura 49). O pior resultado obtido também foi melhor do que a maioria dos resultados anteriores (Figura 50). Em relação aos resultados médios (Figura 48), estes foram melhores do que a maioria das combinações anteriores, com excepção para Pesquisa Tabu / WT e *Simulated Annealing* / L_{max}. Comparando com o sistema MAPS, obteve-se melhores resultados do que a abordagem TCS, mas não tão bons como aqueles obtidos pela abordagem LCS.

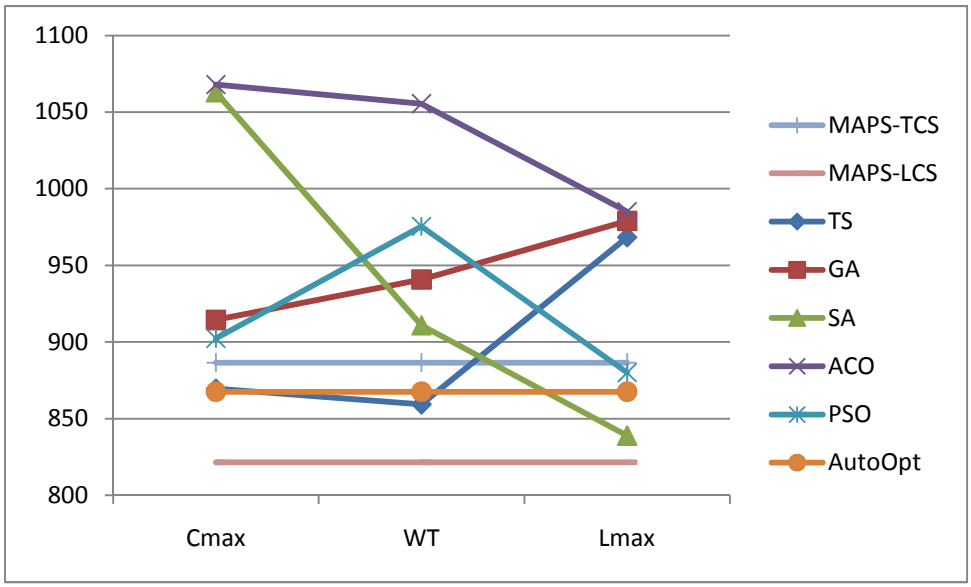

Figura 48 – Gráfico comparativo dos resultados médios obtidos para o problema La02

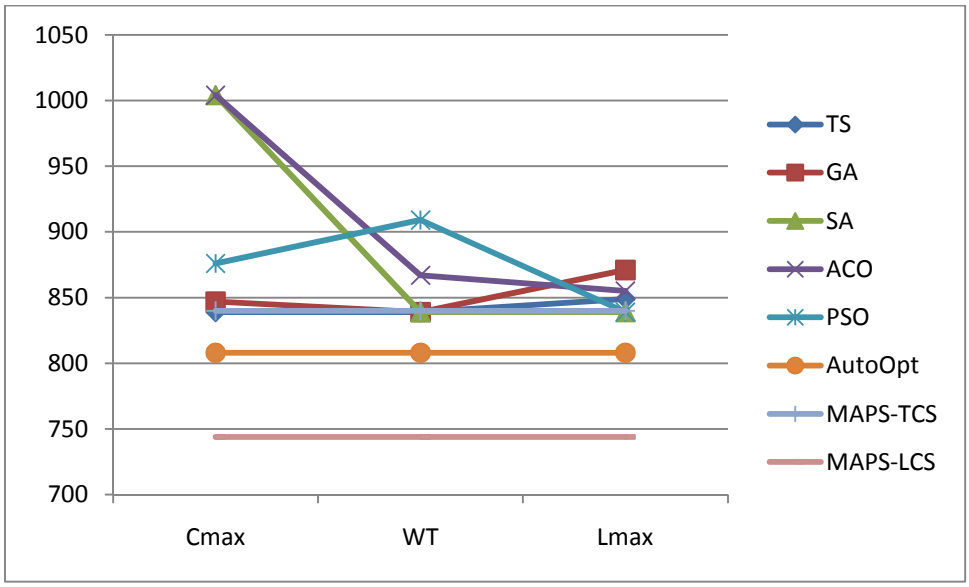

Figura 49 – Gráfico comparativo dos melhores resultados obtidos para o problema La02

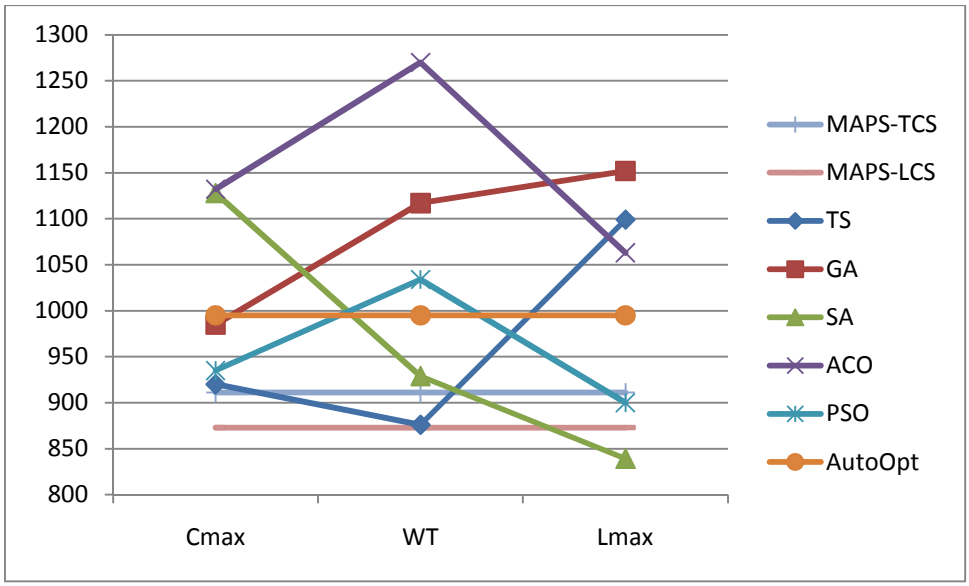

Figura 50 – Gráfico comparativo dos piores resultados obtidos para o problema La02

Para **La03**, também se conseguiu melhorar o melhor resultado, ainda que ligeiramente (Figura 52). Desta vez, obteve-se o valor 747 em contraste com o valor 749 obtido por Pesquisa Tabu / WT. Pela primeira vez, houve uma degradação do pior resultado, em relação à maioria das combinações (Figura 53). O mesmo se verifica nos resultados médios (Figura 51), que só foram melhores do que os resultados obtidos pela Optimização por Colónia de Formigas e algumas outras Meta-heurísticas quando usadas na minimização do atraso máximo (L_{max}). Comparando com o sistema MAPS, os resultados ainda ficaram bastante distantes daqueles obtidos por ambas as abordagens.

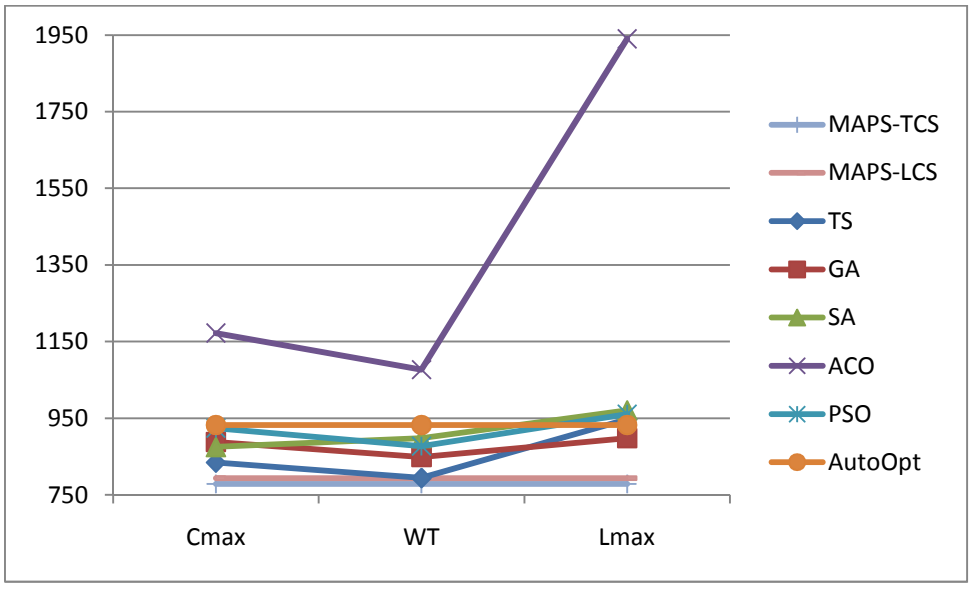

Figura 51 – Gráfico comparativo dos resultados médios obtidos para o problema La03

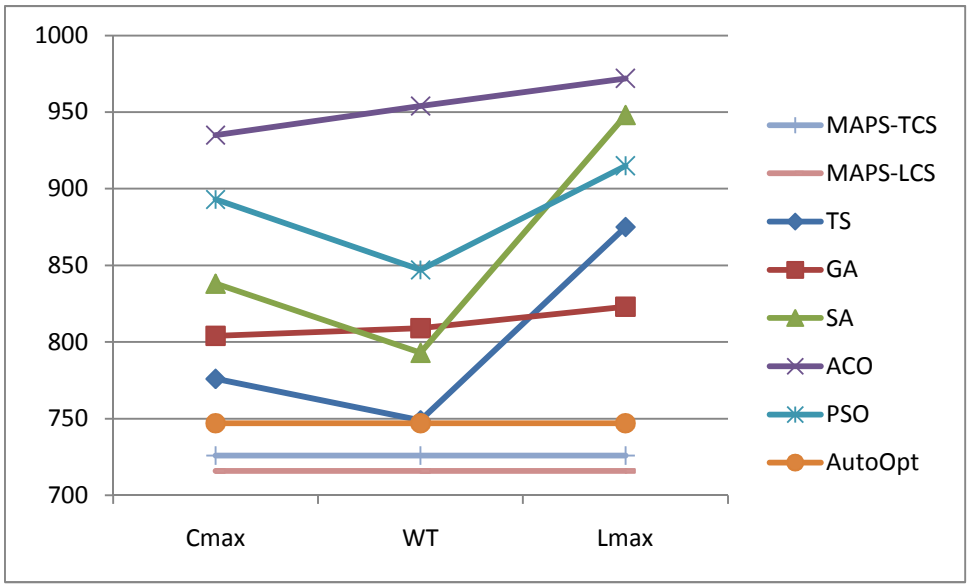

Figura 52 – Gráfico comparativo dos melhores resultados obtidos para o problema La03

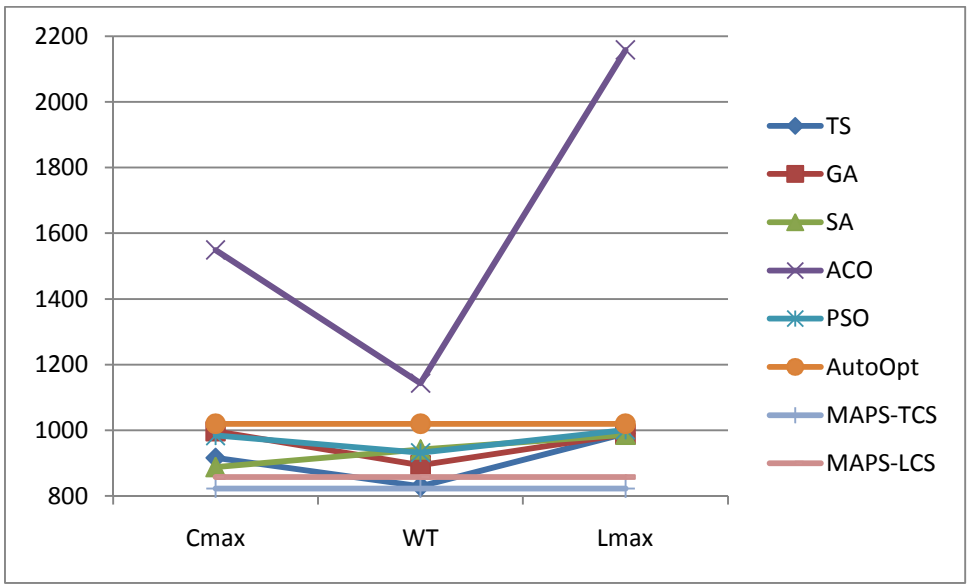

Figura 53 – Gráfico comparativo dos piores resultados obtidos para o problema La03

Finalmente, na **instância La04**, obteve-se um melhor resultado cujo valor é inferior à maioria das combinações anteriores (Figura 55). Apenas a combinação Algoritmos Genéticos / C_{max} obteve melhor resultado do que o módulo de Auto-Optimização, tendo obtido o valor 746 em oposição ao valor 760. O pior resultado foi melhor do que algumas combinações mas também pior em relação a outras (Figura 56). O mesmo se passa em relação aos resultados médios (Figura 54), em que foram melhores do que algumas combinações, mas piores em relação a outras. Comparativamente ao

sistema MAPS, apenas se obteve um melhor resultado do que a abordagem TCS, tendo sido os restantes resultados piores.

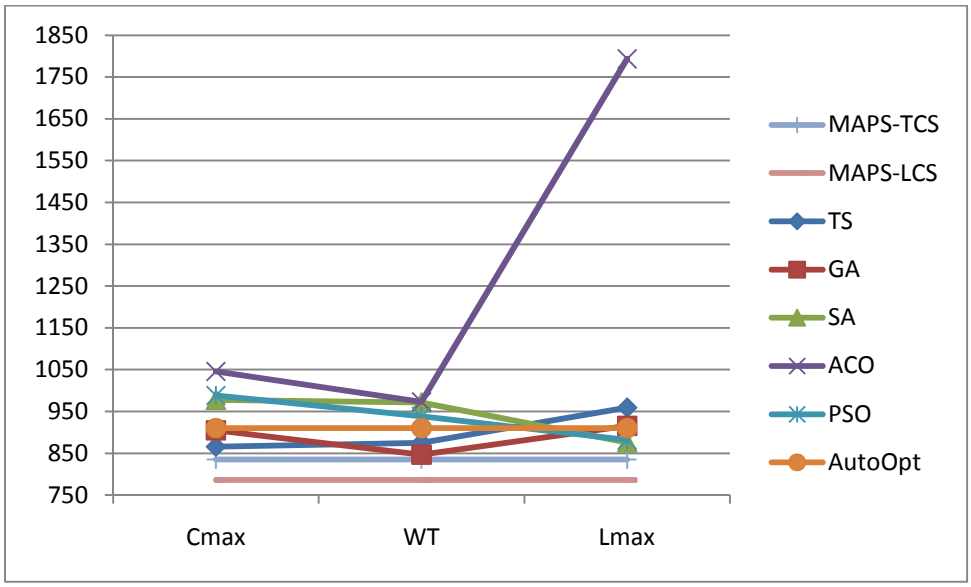

Figura 54 – Gráfico comparativo dos resultados médios obtidos para o problema La04

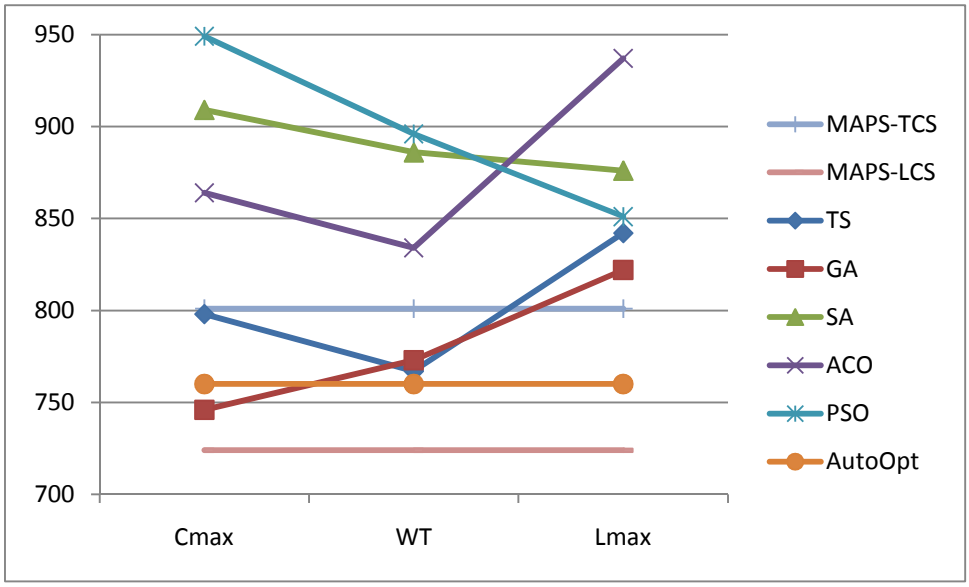

Figura 55 – Gráfico comparativo dos melhores resultados obtidos para o problema La04

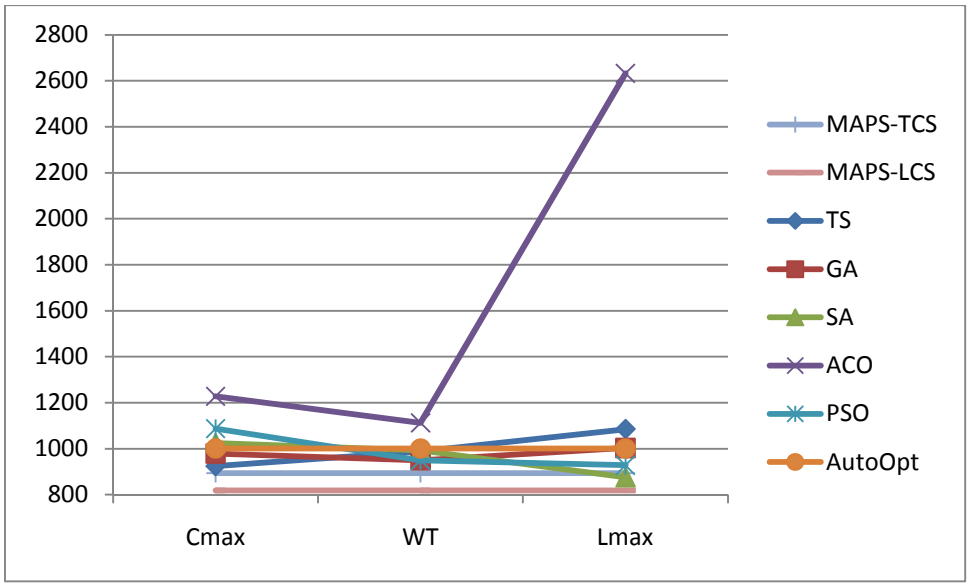

Figura 56 – Gráfico comparativo dos piores resultados obtidos para o problema La04

Para concluir esta análise, é importante salientar a melhoria dos melhores resultados de alguns problemas, a consistência obtida em termos de resultados médios, e o facto dos piores resultados não terem sido degradados, notando-se assim uma vantagem no uso do módulo de Auto-Optimização.

Seguidamente analisar-se-á a distribuição das Meta-heurísticas pelas várias classes de problemas. A Tabela 41 indica, para todas as instâncias, o número de corridas em que cada uma das Meta-heurísticas foi utilizada, estando ilustrada na Figura 57.

Tabela 41 – Número de execuções em que as Meta-heurísticas foram utilizadas, nas várias instâncias de problemas

Prob.	***n***	***m***	**C_{max} Óptimo**	**Pesquisa Tabu**	**Algoritmos Genéticos**	***Simulated Annealing***	**Optimização por Colónia de Formigas**	***Particle Swarm Optimization***
FT06	6	6	55	14	18	11	39	18
FT10	10	10	930	0	0	100	0	0
FT20	20	5	1165	0	0	0	100	0
La01	10	5	666	24	20	11	19	26
La02	10	5	655	17	21	16	16	30
La03	10	5	597	13	26	23	13	25
La04	10	5	590	22	22	12	17	27
Total para os problemas de (Lawrence, 1984)				76	89	62	65	108

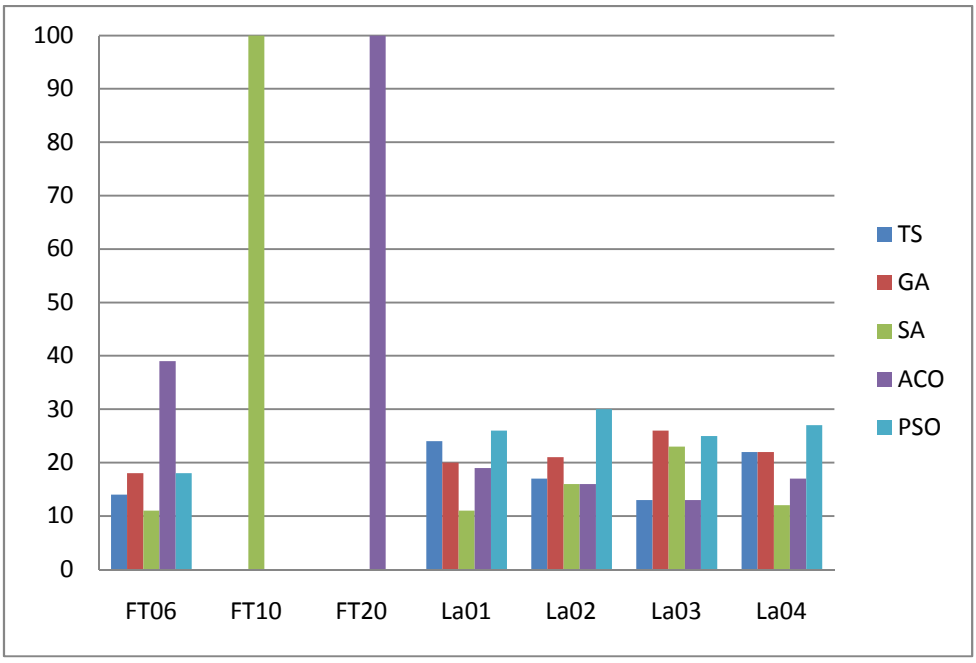

Figura 57 – Gráfico ilustrativo do número de execuções em que as Meta-heurísticas foram utilizadas, nas várias instâncias de problemas

À primeira vista denota-se um equilíbrio na utilização das Meta-heurísticas, com excepção dos problemas FT10 e FT20. Nestes problemas só foi utilizada uma Meta-heurística, devido a razões diferentes:

- A utilização exclusiva do *Simulated Annealing* na **instância FT10** deveu-se ao facto de não existirem casos similares anteriores com uma razão entre o tempo de conclusão óptimo e o tempo de conclusão superior à razão mínima especificada. Assim, o sistema utilizou sempre o melhor caso tendo em conta a sua eficiência-eficácia, como descrito no algoritmo da fase de Reutilização;
- No **FT20** só foi utilizada a Optimização por Colónia de Formigas porque os casos com similaridade superior ou igual à similaridade mínima são apenas os casos resultantes da resolução do mesmo problema, e, entre esses, apenas uma minoria satisfaz o critério da razão mínima, tendo sido todos eles obtidos por intermédio desta Meta-heurística, correspondentes aos melhores resultados obtidos para este problema. Esta situação evidencia que a Optimização por Colónia de Formigas apresenta vantagem na resolução de problemas de grandes dimensões.

No **FT06**, curiosamente a Meta-heurística mais utilizada foi a Optimização por Colónia de Formigas, embora isso não signifique que tenha sido aquela que tirou melhores resultados. Nas restantes técnicas nota-se um equilíbrio significativo.

Nos **problemas de Lawrence** (1984) faz sentido analisar a utilização das Meta-heurísticas conjuntamente, uma vez que são todos problemas similares. Assim, a técnica mais utilizada foi o *Particle Swarm Optimization*, seguida pelos Algoritmos Genéticos e Pesquisa Tabu, e finalmente, uma utilização mais reduzida do *Simulated Annealing* e da Optimização por Colónia de Formigas. Torna-se no entanto importante salientar que isto não significa que as Meta-heurísticas mais utilizadas são aquelas que tiram melhores resultados, como se pode ver de seguida, embora seja possível concluir que são importantes para os resultados médios.

Na Tabela 42 são apresentadas as parametrizações das Meta-heurísticas usadas na obtenção dos melhores resultados, para todos os problemas.

Tabela 42 – Parametrizações usadas na obtenção dos melhores resultados de todas as instâncias

Prob.	MH	Parâmetros	Valores
FT06	Algoritmos Genéticos	Solução Inicial	SeqNivel
		Função Objectivo	L_{max}
		Número de gerações	98
		Afastamento	10%
		Tamanho da população	100%
		Taxa de cruzamento	75%
		Taxa de mutação	1%
FT10	*Simulated Annealing*	Solução Inicial	SeqNivel
		Função Objectivo	C_{max}
		Critério de paragem	38
		Número de iterações à mesma temperatura	14
		Temperatura inicial	15
		Factor de redução da temperatura (*alpha*)	80%
FT20	Optimização por Colónia de Formigas	Solução Inicial	SeqNivel
		Função Objectivo	L_{max}
		Critério de paragem	250
		Taxa de evaporação	80%
		Número de formigas por colónia	50
		Número de colónias	1
		Alpha	1
		Beta	1
La01	Optimização por Colónia de Formigas	Solução Inicial	SeqNivel
		Função Objectivo	WT
		Critério de paragem	103
		Taxa de evaporação	80%
		Número de formigas por colónia	24
		Número de colónias	1
		Alpha	1
		Beta	1

La02	*Simulated Annealing*	Solução Inicial	SeqNivel
		Função Objectivo	C_{max}
		Critério de paragem	33
		Número de iterações à mesma temperatura	16
		Temperatura inicial	15
		Factor de redução da temperatura (*alpha*)	80%
La03	Algoritmos Genéticos	Solução Inicial	SeqNivel
		Função Objectivo	C_{max}
		Número de gerações	92
		Afastamento	10%
		Tamanho da população	100%
		Taxa de cruzamento	75%
		Taxa de mutação	1%
La04	*Simulated Annealing*	Solução Inicial	SeqNivel
		Função Objectivo	C_{max}
		Critério de paragem	38
		Número de iterações à mesma temperatura	16
		Temperatura inicial	15
		Factor de redução da temperatura (*alpha*)	80%

Na **instância FT06**, tal como se verifica, apesar da Optimização por Colónia de Formigas ter sido a Meta-heurística mais vezes utilizada, o melhor resultado foi obtido pelos Algoritmos Genéticos. Uma vez que existem vários casos com o mesmo tempo de conclusão, a escolha do melhor resultado foi decidida através dos tempos de execução. Uma curiosidade relativamente à função objectivo utilizada, L_{max}, que tão maus resultados tinha nos resultados prévios, conseguiu tirar um bom resultado.

Na **instância FT10**, como já se tinha verificado, os resultados foram obtidos pelo uso do *Simulated Annealing*, sendo o melhor resultado obtido com um critério de paragem ligeiramente mais elevado, embora com menos uma iteração à mesma temperatura, relativamente às parametrizações iniciais.

Na **instância FT20** foi usada uma parametrização igual à utilizada nos resultados obtidos anteriormente, tendo sido utilizada a função objectivo L_{max}.

Tal como já se tinha referido, nos problemas de Lawrence (1984), embora o *Particle Swarm Optimization* tenha sido a Meta-heurística mais utilizada, os melhores resultados foram obtidos através da utilização de outras Meta-heurísticas. Na **instância La01** foi utilizada a Optimização por Colónia de Formigas com WT, com mais 3 iterações no critério de paragem e menos uma formiga por colónia. Nas **instâncias La02 e La04**, foi utilizado *Simulated Annealing* com critério de paragem superior e mais uma iteração à mesma temperatura, utilizando a função objectivo C_{max}. Na **instância**

La03, foram utilizados os Algoritmos Genéticos, com menos 8 gerações do que a parametrização inicial.

Como conclusão note-se que com pequenas mudanças nas parametrizações se conseguiram bons resultados. Estas pequenas mudanças são resultantes do facto dos casos terem grande similaridade entre si, e como tal, é usado pouco crédito na atribuição de perturbação. Note-se também a utilização significativa do critério de minimização C_{max} na obtenção dos melhores resultados.

6.3.3. Outros resultados e conclusões

Nesta secção são apresentados outros resultados considerados relevantes para o estudo computacional. Primeiro é analisado um problema *Job-Shop* estático que não tinha sido testado anteriormente, e seguidamente é analisado um problema de *Job-Shop* Alargado Dinâmico.

6.3.3.1. Problema *Job-Shop* Estático

O problema considerado para este estudo foi o *La05* de Lawrence (1984). Primeiro, sem módulo de Auto-Optimização, foram retirados resultados médios, melhores e piores de 5 execuções, para todas as Meta-heurísticas e, em cada uma, para as diferentes Funções Objectivo. Estes resultados são mostrados na Tabela 43 e na Tabela 44. As parametrizações usadas para as Meta-heurísticas são iguais às apresentadas na Tabela 37.

De seguida foi activado o módulo de Auto-Optimização e executado o problema igualmente 5 vezes, tendo sido tirados resultados médios, melhores e piores, apresentados na Tabela 45. A base de casos de suporte a estes resultados foi a resultante do estudo computacional anterior, da secção 6.3.2, com 805 casos (Tabela 45).

Comparando os tempos de conclusão, nota-se que o tempo de conclusão óptimo foi atingido, tal como tinha sido com o uso do *Simulated Annealing* com WT e *Particle Swarm Optimization* com C_{max} e WT. A nível de resultados médios, estes foram melhores do que a maioria das combinações Meta-heurística / função objectivo, com excepção de Pesquisa Tabu / WT, *Simulated Annealing* / WT, e *Particle Swarm Optimization* / WT. Note-se que com a minimização dos atrasos pesados (WT) se obteve as soluções com os melhores tempos de conclusão. Por último, em relação aos piores resultados, estes foram melhores do que a generalidade, notando-se também melhorias a esse nível.

Tabela 43 – Resultados da Pesquisa Tabu, dos Algoritmos Genéticos e do *Simulated Annealing* na resolução da instância La05

Prob.	*n*	*m*	C_{max} Óptimo	F. Obj.	Pesquisa Tabu						Algoritmos Genéticos						*Simulated Annealing*					
					C_{max}			TimeExec (s)			C_{max}			TimeExec (s)			C_{max}			TimeExec (s)		
					Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior
La05	10	5	593	C_{max}	669.61	599	806	29.82	27.74	30.86	661.21	607	734	7.25	7.03	7.87	636.61	610	671	0.34	0.22	0.44
				WT	620	605	639	28.44	28.19	28.61	715	634	793	7.39	7.12	7.90	611.61	593	619	0.21	0.19	0.22
				L_{max}	838	731	876	29.12	28.76	29.80	630	623	673	7.58	7.07	7.89	636	636	636	0.19	0.19	0.20

Tabela 44 – Resultados da Optimização por Colónia de Formigas e do *Particle Swarm Optimization* na resolução da instância La05

Prob.	*n*	*m*	C_{max} Óptimo	F. Obj.	Optimização por Colónia de Formigas						*Particle Swarm Optimization*					
					C_{max}			TimeExec (s)			C_{max}			TimeExec (s)		
					Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior	Médio	Melhor	Pior
La05	10	5	593	C_{max}	830.8	664	990	1.26	1.21	1.35	718.4	670	764	1.22	1.0	1.46
				WT	670.61	629	706	1.43	1.4	1.44	616.61	593	639	1.33	1.29	1.38
				L_{max}	879.15	860	972	1.41	1.38	1.42	632.8	593	669	1.20	1.04	1.34

Tabela 45 – Resultados do módulo de Auto-Optimização na resolução da instância La05

Prob.	*n*	*m*	C_{max} Óptimo	C_{max}			TimeExec (s)		
				Médio	Melhor	Pior	Médio	Melhor	Pior
La05	6	6	593	626.21	593	668	26.43	26.3	27.21

A parametrização usada para a obtenção do melhor caso é apresentada na Tabela 46. Saliente-se que é a primeira vez que a Pesquisa Tabu é usada na obtenção de um melhor resultado, em problemas de Lawrence, e, de forma similar às outras Meta-heurísticas, existe apenas uma pequena mudança no critério de paragem, tendo sido feito aqui um decréscimo do valor, resultante da pequena perturbação resultante de grande similaridade entre casos.

Tabela 46 – Parametrização do melhor caso na resolução da instância La05

Prob.	MH	Parâmetros	Valores
La05	Pesquisa Tabu	Solução Inicial	SeqNivel
		Função Objectivo	WT
		Critério de paragem	92
		Afastamento	10%
		Subvizinhança	100%
		Tamanho da Lista Tabu	1

6.3.3.2. Problema *Job-Shop* Alargado Dinâmico

Nesta subsecção é analisado o desempenho do sistema AutoDynAgents integrado com o módulo de Auto-Optimização na resolução de um problema *Job-Shop* Alargado Dinâmico. Este problema é composto por 7 tarefas a serem processadas em 8 máquinas, e corresponde a um produto composto definido em (Madureira, 2003).

Na Tabela 47 estão representados os dados das tarefas a processar neste problema dinâmico. É possível reparar em prioridades diferentes assim como datas de lançamento e datas de entrega diferentes, características dos problemas *Job-Shop* Alargado (Madureira, 2003).

Tabela 47 – Dados das tarefas a processar no problema *Job-Shop* Alargado Dinâmico

Tarefas	Produtos	Prioridade	Data de Lançamento	Data de Entrega
J1	P1	3	0	55
J2	P2	2	2	70
J3	P3	2	1	85
J4	P4	1	5	65
J5	P5	3	0	72
J6	P1	1	6	85
J7	P3	1	6	98

Na Tabela 48 estão identificados os tipos de eventos ocorridos neste problema dinâmico. No instante 5 houve um cancelamento da tarefa J4, no instante 12 foram alteradas as datas de lançamento e entrega da tarefa J6, no instante 15 foi alterado o peso da tarefa J7 e no instante 30 ocorreu um evento de chegada de tarefas, a ser introduzida a tarefa J8.

Tabela 48 – Lista de eventos ocorridos no problema *Job-Shop* Alargado Dinâmico

Instante	Tipo de Evento	Tarefas
5	Cancelamento de tarefas	J4
12	Alteração de datas de lançamento e de entrega	J6: dt_lan=28; dt_ent=88
15	Alteração de prioridade	J7: peso=3
30	Chegada de tarefas	J8: peso=1; dt_lan=32; dt_ent=87

A tarefa J8 tem uma estrutura igual à tarefa J1, em termos de gama operatória, sendo diferente o seu peso, data de lançamento e data de entrega.

Inicialmente procedeu-se à execução "off-line" do problema, tendo-se obtido um plano inicial. A partir deste plano inicial foram executados os eventos e chegou-se a um plano final. Os resultados obtidos do plano inicial e do plano final estão apresentados na Tabela 49, bem como as Meta-heurísticas e funções objectivo usadas na resolução dos vários eventos que ocorreram.

Tabela 49 – Resultados obtidos para o problema *Job-Shop* Alargado Dinâmico

	MH	F. Obj	Cmax	TimeExec
Plano Inicial	ACO	C_{max}	149	6.21
1º Evento	SA	C_{max}	-	-
2º Evento	SA	WT	-	-
3º Evento	TS	WT	-	-
Plano Final	GA	C_{max}	165	7.14

Analisando os resultados, nota-se que o módulo de Auto-Optimização funciona de igual forma com dinamismo como sem dinamismo. Sempre que um evento ocorre, é como se um novo caso surgisse no sistema, permitindo que a parametrização mude em tempo de execução.

As parametrizações usadas pelas diversas Meta-heurísticas estão descritas na Tabela 50.

Tabela 50 – Meta-heurísticas e respectivas parametrizações usadas no problema *Job-Shop* Alargado Dinâmico

	MH	Parâmetros	Valores
Plano Inicial	ACO	Solução Inicial	SeqNivel
		Função Objectivo	C_{max}
		Critério de paragem	47
		Taxa de evaporação	0.80
		Número de formigas por colónia	15
		Número de colónias	1
		Alpha	1

		Beta	1
1º Evento	SA	Solução Inicial	SeqNivel
		Função Objectivo	C_{max}
		Critério de paragem	31
		Número de iterações à mesma temperatura	12
		Temperatura inicial	15
		Factor de redução da temperatura (*alpha*)	0.8
2º Evento	SA	Solução Inicial	SeqNivel
		Função Objectivo	WT
		Critério de paragem	29
		Número de iterações à mesma temperatura	11
		Temperatura inicial	15
		Factor de redução da temperatura (*alpha*)	0.8
3º Evento	TS	Solução Inicial	SeqNivel
		Função Objectivo	WT
		Critério de paragem	102
		Afastamento	0.10
		Subvizinhança	1.00
		Tamanho da Lista Tabu	1
Plano Final	GA	Solução Inicial	SeqNivel
		Função Objectivo	C_{max}
		Número de gerações	113
		Afastamento	0.10
		Tamanho da população	1.00
		Taxa de cruzamento	0.75
		Taxa de mutação	0.01

Como conclusão, salienta-se o facto do módulo de Auto-Optimização funcionar de forma idêntica, e apresentando desempenho similar, em ambientes dinâmicos como funciona em ambientes estáticos.

6.4. Sumário

Neste capítulo foi apresentado o estudo computacional efectuado para a validação e análise do sistema AutoDynAgents com a incorporação do módulo de Auto-Optimização implementado no âmbito deste trabalho de mestrado. Os problemas usados para a obtenção de resultados foram os problemas académicos de Fisher e Thompson (1963) e de Lawrence (1984).

O estudo computacional encontra-se dividido em duas partes principais, sendo que, na primeira, foi efectuada uma inicialização da Base de Casos, tendo sido inseridos, para cada problema, os

melhores resultados obtidos anteriormente com o sistema AutoDynAgents, usando as diversas Meta-heurísticas implementadas e, para cada uma delas, os valores para os critérios de optimização C_{max} (tempo total de conclusão), WT (atrasos pesados) e L_{max} (atraso total).

Na segunda parte do estudo computacional foram apresentados os resultados computacionais obtidos pelo sistema AutoDynAgents. Primeiro apresentaram-se os resultados que serviram de comparação com os resultados obtidos, estando aqui incluídos os resultados previamente obtidos pelo sistema AutoDynAgents, sem o módulo de Auto-Optimização. Seguidamente foram apresentados os resultados obtidos pelo sistema, já com o módulo de Auto-Optimização incluído, tendo sido tiradas algumas conclusões sobre os mesmos. Verificou-se vantagem na utilização do módulo de Auto-Optimização, uma vez que se conseguiu melhorar alguns dos melhores resultados obtidos bem como melhorar a consistência dos resultados médios, na maioria dos problemas testados, comparativamente à maioria das combinações Meta-heurística / função objectivo.

Por último, são apresentados outros resultados relevantes e respectivas conclusões, nomeadamente na resolução de um outro problema *Job-Shop* estático e a execução de um problema *Job-Shop* Alargado dinâmico.

CAPÍTULO 7. CONCLUSÃO

7.1. Resumo

Nesta dissertação começou-se por estudar os problemas de Optimização Combinatória, em especial o problema do Escalonamento *Job-Shop* e respectivas abordagens de resolução. Estas abordagens recaem essencialmente nas Meta-heurísticas, técnicas com o objectivo de guiar e melhorar a pesquisa de soluções de modo a superar os óptimos locais, tendo sido descritas principalmente as usadas no sistema AutoDynAgents, isto é, a Pesquisa Tabu, o *Simulated Annealing*, os Algoritmos Genéticos, a Optimização por Colónia de Formigas, e o *Particle Swarm Optimization*.

Uma vez que o sistema AutoDynAgents é um Sistema Multi-Agente, foi abordado o conceito de agente, suas características e limitações, passando-se para os sistemas baseados em agentes, onde se descreveram os vários modelos descritos na literatura, desde os modelos hierárquicos, passando pelos modelos em equipa, até chegar aos modelos de mercado. Descreveu-se de seguida o Escalonamento Dinâmico e as arquitecturas para Escalonamento baseadas em Agentes, nomeadamente as Arquitecturas Autónomas e as Arquitecturas com Mediador.

Sendo o AutoDynAgents um sistema autónomo, foi importante estudar o conceito de Computação Autónoma, tendo sido feita uma descrição dos principais conceitos e componentes da sua arquitectura.

Em relação ao tema principal desta tese, a Aprendizagem, foi abordada a Aprendizagem Automática, com uma descrição sucinta das suas subáreas, a Aprendizagem Supervisionada, a Aprendizagem Não-Supervisionada, a Aprendizagem Semi-Supervisionada, e a Aprendizagem por Reforço. Foi descrito com maior detalhe o Raciocínio baseado em Casos, uma vez que foi importante para o desenvolvimento do módulo de Auto-Optimização, tendo sido dado alguns exemplos de aplicação em Escalonamento.

Continuando no tema da Aprendizagem, este foi abordado para Sistemas Multi-Agente, principalmente para Sistemas de Fabrico, tendo sido descritos os principais aspectos necessários de aprendizagem. Foi também efectuada uma análise aos aspectos de aprendizagem de agentes, tendo sido descrita a Aprendizagem de Agente Único, em oposição à Aprendizagem Multi-Agente, bem como descritos os métodos de Aprendizagem On-line e Off-line. Por fim, foram descritas algumas técnicas de aprendizagem, nomeadamente a Aprendizagem em Sistemas Reactivos, a Aprendizagem Social, a Aprendizagem em Equipa, e a Aprendizagem Concorrente.

Passando para a parte mais prática desta tese, foi descrito o sistema sobre o qual foi implementado o módulo de Auto-Optimização, o sistema AutoDynAgents, que consiste num sistema

de escalonamento autónomo, no qual uma comunidade de agentes modela um sistema real de fabrico sujeito a perturbações, para a resolução distribuída, cooperativa e autónoma de problemas de escalonamento. Sobre este sistema, foi descrita a sua arquitectura, com os principais tipos de agentes, bem como os principais módulos, nomeadamente o módulo de Escalonamento e o módulo de Adaptação Dinâmica.

Integrado no sistema AutoDynAgents, foi desenvolvido o módulo de Auto-Optimização, cujo objectivo é dotar o sistema de capacidades de aprendizagem, através do uso de experiência para a resolução de novos problemas. Para isto foi usado um sistema de Raciocínio baseado em Casos para a escolha de uma Meta-heurística, e respectivos parâmetros de entrada, na resolução de problemas de escalonamento. Esta auto-parametrização tem por base casos passados, em que foram resolvidos problemas de escalonamento com sucesso. Assim, foi descrita a arquitectura do módulo desenvolvido, contendo uma descrição da Base de Casos bem como de todas as fases do ciclo do Raciocínio baseado em Casos, tendo sido descritos igualmente os principais algoritmos. Para melhor se entender o funcionamento do módulo de Auto-Optimização, este foi ilustrado com um exemplo.

Para se validar o desenvolvimento do módulo de Auto-Optimização, foi efectuado um estudo computacional, composto por várias fases. Em primeiro lugar, procedeu-se à inicialização da Base de Casos com definição dos casos inseridos e respectivas parametrizações iniciais. Estes dados tiveram por base os resultados prévios obtidos pelo sistema AutoDynAgents, sem o módulo de Auto-Optimização incluído. Seguidamente, compararam-se os resultados obtidos pela inclusão do módulo desenvolvido com os resultados prévios e com os resultados do sistema MAPS (Wellner & Dilger, 1999). Para finalizar o estudo computacional, apresentou-se a resolução de um novo problema *Job-Shop* estático e a resolução de um problema *Job-Shop* Alargado dinâmico, em que se confirmou a consistência do módulo de Auto-Optimização, mesmo em problemas dinâmicos.

7.2. Objectivos alcançados

O primeiro objectivo consistiu na revisão do estado da arte das abordagens de resolução de problemas de Optimização Combinatória, principalmente do problema de Escalonamento, tendo sido estudadas algumas Meta-heurísticas, nomeadamente a Pesquisa Tabu, o *Simulated Annealing*, os Algoritmos Genéticos, a Optimização por Colónia de Formigas, e o *Particle Swarm Optimization*.

Também se pretendia uma revisão do estado da arte de aprendizagem em Sistemas Multi-Agente, onde se começou por estudar a Aprendizagem Automática e suas subáreas, tendo sido abordado principalmente o Raciocínio baseado em Casos, que se enquadra na Aprendizagem baseada em Instâncias. Foram também referidos os aspectos de aprendizagem em Sistemas de Fabrico, seguindo-se a relação entre a Aprendizagem Automática e os Sistemas Multi-Agente, e os aspectos de aprendizagem de agentes, onde se comparou a Aprendizagem de Agente Único com a

Aprendizagem Multi-Agente, bem como métodos de Aprendizagem On-line e Off-line. Para finalizar esta revisão de estado da arte, estudou-se e descreveu-se várias técnicas de aprendizagem referidas na literatura, nomeadamente a Aprendizagem em Sistemas Reactivos, a Aprendizagem Social, a Aprendizagem em Equipa e a Aprendizagem Concorrente.

O terceiro e principal objectivo passou pela proposta e desenvolvimento de um módulo de Auto-Optimização para integração no sistema AutoDynAgents, para a resolução de novos problemas de Escalonamento, com base em experiência anterior. Assim, o módulo desenvolvido baseou-se em Raciocínio baseado em Casos, para escolha e parametrização de Meta-heurísticas, na resolução de novos problemas de escalonamento, requerendo o mínimo de intervenção humana, apenas necessária na especificação das parametrizações iniciais quando os novos problemas não encontram casos anteriores similares.

O último objectivo consistiu na realização de um estudo computacional do sistema AutoDynAgents com o módulo de Auto-Optimização incluído, comparando-se resultados obtidos com resultados anteriores e com resultados de um sistema referido na literatura (MAPS de Wellner e Dilger (1999)). Através desta análise comparativa chegou-se à conclusão de que o uso do módulo de Auto-Optimização constitui realmente uma vantagem, uma vez que se conseguiu melhorar os melhores resultados obtidos, bem como se conseguiu manter uma consistência elevada dos resultados médios. São notórias também algumas melhorias das soluções ao longo do tempo, uma vez que o módulo de Auto-Optimização vai ficando cada vez mais afinado e preciso.

7.3. Limitações e trabalho futuro

Na realização deste trabalho foram detectadas algumas limitações e vulnerabilidades subjacentes ao desenvolvimento e implementação do módulo de Auto-Optimização. Uma dessas limitações é o facto de apenas se ter considerado a heurística *SeqNivel* para a geração da solução inicial das Meta-heurísticas nos casos de inicialização da Base de Casos. 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 *SeqNivel* é ou não a mais adequada, dependendo da evolução das parametrizações das Meta-heurísticas na resolução de problemas de escalonamento diferentes.

Outro aspecto 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, levando a um decréscimo do desempenho do módulo de Auto-Optimização. 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 detectar grupos de casos similares, para ser possível eliminar os

piores casos, uma vez 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-Optimização como as Meta-heurísticas em si contêm alguma aleatoriedade, o que pode levar a bons resultados, mas também a resultados menos bons.

De modo a superar as limitações apontadas, 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.

Ainda a nível de trabalho futuro, considera-se também:

- A implementação do módulo de Auto-Optimização seguindo uma abordagem de Aprendizagem Concorrente, em que cada AgenteRecurso tem o seu próprio módulo, optimizando os seus problemas de máquina única, de modo a ser possível comparar com a abordagem de Aprendizagem em Equipa implementada;
- Um estudo aprofundado dos parâmetros dos algoritmos implementados nas diversas fases do Raciocínio baseado em Casos, de modo a ser possível concluir quais os valores mais adequados a cada um;
- A implementação de outras técnicas de aprendizagem, para se contrastar com o Raciocínio baseado em Casos, como, por exemplo, Redes Neuronais e Lógica Difusa.

7.4. Considerações finais

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

Também se gostaria de referir que, no âmbito do sistema AutoDynAgents e do módulo de Auto-Optimização, foram publicados alguns artigos, nomeadamente (Madureira et al., 2008), (Madureira et al., 2008b), e (Madureira et al., 2009b).

BIBLIOGRAFIA

Aamodt, A. & Plaza, E., 1994. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1), pp.39-59.

Adams, J., Balas, E. & Zawack, D., 1988. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 34(3).

Alonso, E. et al., 2001. Learning in multi-agent systems. *The Knowledge Engineering Review*, 16(3), pp.277-84.

Alpaydin, E., 2004. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Applegate, D.L., Bixby, R.E., Chvátal, V. & Cook, W.J., 2006. *The Traveling Salesman Problem*. Princeton University Press.

Aydin, M.E. & Öztemel, E., 2000. Job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2-3), pp.169-78.

Bäck, T., 1996. Evolutionary Algorithms in Theory and Practice: Evolutionary Strategies. In *Evolutionary Programming, and Genetic Algorithms*. Oxford Press.

Baker, K., 1974. *Introduction to sequencing and scheduling*. New York: Wiley.

Barto, A., Sutton, R. & Watkins, C., 1990. Learning and sequential decision making. In Gabriel, M. & Moore, J., eds. *Learning and computational neuroscience: foundations of adaptive networks*. Cambridge, 1990. MIT Press.

Beddoe, G. & 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*, pp.649-71.

Beddoe, G., Petrovic, S. & Li, J., 2009. A hybrid metaheuristic case-based reasoning system for nurse rostering. *Journal of Schedling 12*, 2 April. pp.99-119.

Bernon, C., Cossentino, M. & Pavón, J., 2005. An Overview of Current Trends in European AOSE Research. In *Informatica 29*., 2005.

Bezek, A., Gams, M. & Bratko, I., 2006. Multi-agent strategic modeling in a robotic soccer domain. In *Proceedings of the Fifth international Joint Conference on Autonomous Agents and Multiagent Systems*. Hakodate, Japan, 2006.

Blum, C. & Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. 35*, September. pp.268-308.

Bodenhofer, U., 2004. Genetic Algorithms: Theory and Applications. *Lecture notes, Fuzzy Logic Laboratorium Linz-Hagenberg*.

Bongaerts, L., Monostori, L., McFarlane, D. & Kádár, B., 2000. Hierarchy in distributed shop floor control. *Computers in Industry*, 43(2), pp.123-37.

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

Brustolini, J., 1991. *Autonomous Agents: characterization and requirements*. Carnegie Mellon Technical Report CMU-CS-91-204. Pittsburgh.

Bull, L. & Holland, O., 1997. Evolutionary computing in multiagent environments: Eusociality. In *Proceedings of Seventh Annual Conference on Genetic Algorithms*., 1997.

Burgard, W., Moors, M., Stachniss, C. & Schneider, F., 2005. Coordinated Multi-Robot Exploration. *IEEE Transactions on Robotics*, 21, pp.376--386.

Burke, E.K., MacCarthy, B.L., Petrovic, S. & Qu, R., 2002. Knowledge Discovery in a Hyper-Heuristic for Course Timetabling Using Case-Based Reasoning. In *PATAT 2002*., 2002.

Carbonell, J. et al., 1991. PRODIGY: an integrated architecture for planning and learning. *SIGART Bull.*, 2(4), pp.51-55.

Cerny, V., 1985. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *J. Optim. Theory Appl. 45*, pp.41-51.

Coelho, H., 1994. *Inteligência Artificial em 25 lições*. Fundação Calouste Gulbenkian.

Coello, J.M.A. & Santos, R.C., 1999. Integrating CBR and heuristic search for learning and reusing solutions in real-time task scheduling. In *Proceedings of 3rd International Conference on Case-Based Reasoning*. Germany, 1999. Springer-Verlag.

Cowling, P., Kendall, G. & Soubeiga, E., 2000. Hyperheuristic Approach to Scheduling a Sales Summit. In *Selected papers of Proceedings of the Third International Conference of Practice and Theory of Automated Timetabling (PATAT2000)*., 2000. Springer.

Cunningham, P. & Smyth, B., 1997. Case-Based Reasoning in Scheduling: Reusing Solution Components. *The International Journal of Production Research*, 35, pp.2947--2961.

Darwin, C., 1859. *On the Origins of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle of Life*.

Dasgupta, D., 1999. *Artificial Immune Systems and Their Applications*. Springer-Verlag.

Davis, R., 1980. Report on the Workshop on Distributed Artificial Intelligence. In *SIGART Newsletter*., 1980.

Davis, L., 1991. *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

Dell'Amico, M. & Trubian, M., 1993. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41(1-4), p.231–252.

Dietterich, T.G., 2003. Machine Learning. In *Nature Encyclopedia of Cognitive Science*. London, 2003.

Dorigo, M., Maniezzo, V. & Colorni, A., 1991. *The ant system: an autocatalytic optimizing*. Technical Report TR91-016. Politecnico di Milano.

Fang, H., Ross, P. & Corne, D., 1993. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *Proceedings of the Fifth International Conference on Genetic Algorithms*., 1993. Morgan Kaufmann.

Fang, H.L., Ross, P. & Corne, D., 1994. A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems. In *The 11th European Conference on Artificial Intelligence (ECAI'94)*., 1994. John Wiley & Sons, Ltd.

Feo, T.A. & Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8, pp.67-71.

Fisher, H. & Thompson, G.L., 1963. Probabilistic learning combinations of local Job Shop scheduling rules. In *Industrial Scheduling*. Prentice Hall. pp.225-51.

Franklin, S. & Graesser, A., 1996. Is it an Agent or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96)*. Berlin, Germany, 1996. Springer-Verlag.

Ganek, A., 2006. Overview of Autonomic Computing: Origins, Evolution, Direction. In *Autonomic Computing – Concepts, Infrastructure, and Applications*.

Garland, A. & Alterman, R., 2004. Autonomous Agents that Learn to Better Coordinate. *Autonomous Agents and Multi-Agent Systems*, 8(3), pp.267-301.

Geem, Z.-W., 2000. *Optimal Design of Water Distribution Networks*. Doctoral dissertation. Korea University.

Gentner, D., 1983. Structure mapping - a theoretical framework for analogy. *Cognitive Science*, 7, pp.155-70.

Glover, F., 1986. Future paths for integer prog. and links to artificial intelligence. *Comp. & Ops. Res.*, 5, pp.533-49.

Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Goldman, C.V. & Rosenschein, J.S., 1996. Mutually Supervised Learning in Multiagent Systems. In *IJCAI '95: Proceedings of the Workshop on Adaption and Learning in Multi-Agent Systems*. London, UK, 1996. Springer-Verlag.

Gorodetsky, V., Karsaev, O., Samoylov, V. & Skormin, V., 2008. Multi-Agent Technology for Air Traffic Control and Incident Management in Airport Airspace. In *Proceedings of AAMAS International Workshop Agents in Traffic and Transportation*. Lisboa, Portugal, 2008.

Grolimund, S. & Ganascia, J.-G., 1997. Driving Tabu Search with case-based reasoning. *European Journal of Operational Research*, 103(2), pp.326-38.

Haipeng, Z., Mitsuo, G., Shigeru, F. & Woo, K.K., 2004. Hybrid Ant Colony Optimization for Job-shop Scheduling Problem. *Faji Shisutemu Shinpojiumu Koen Ronbunshu*, 20, pp.304-05.

Hart, E., Ross, P. & Nelson, J., 1998. Solving a Real-world Problem Using an Evolving Heuristically Driven Schedule. *Evolutionary Computation*, 6, pp.61-80.

Holland, J.H., 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan.

Horling, B. & Lesser, V., 2004. A survey of multi-agent organizational paradigms. *Knowl. Eng. Rev.*, 19(4), pp.281-316.

Horn, P., 2001. Autonomic Computing: IBM's Perspective on the State of Information Technology., 2001. IBM Research.

Hu, J. & Wellman, M.P., 1998. Multiagent Reinforcement Learning: Theoret ical Framework and an Algorithm. In *Proc. 15th International Conf. on Machine Learning*., 1998.

IBM, 2005. *An architectural blueprint for autonomic computing*. White Paper.

Jansen, T. & Wiegand, R.P., 2003. Exploring the explorative advantage of the cooperative coevolutionary (1+1) EA. In Cantúú-Paz, E. et al., eds. *Genetic and Evolutionary Computation - GECCO 2003*. Chicago, 2003. Springer-Verlag.

Jascanu, N., 2008. Emotionally based multi-agent e-commerce platform. In *Proceedings of the 7th international Joint Conference on Autonomous Agents and Multiagent Systems: Doctoral Mentoring Program*. Estoril, Portugal, 2008.

Jong, K.D., 2005. *Evolutionary Computation: A unified approach*. MIT Press.

Kaelbling, L., Littman, M. & Moore, A., 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, pp.237-85.

Kennedy, J. & Eberhart, R., 1995. Particle swarm optimization. In *Procedures of the IEEE International Conference on Neural Networks*., 1995.

Kephart, J. & Chess, D., 2003. The Vision of Autonomic Computing. *Computer Magazine*, January.

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P., 1983. Optimization by simulated annealing. *Science*, 13 Maio. p.671–680.

Kolodner, J.L., 1993. *Case-Based Reasoning*. Morgan Kaufmann Publishers Inc.

Kose, H. et al., 2005. Market-Driven Multi-Agent Collaboration in Robot Soccer Domain. In *Cutting Edge Robotics*. Germany, 2005.

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

Kuyer, L., Whiteson, S., Bakker, B. & Vlassis, N., 2008. Multiagent Reinforcement Learning for Urban Traffic Control using Coordination Graphs. In *ECML 2008: Proceedings of the Nineteenth European Conference on Machine Learning*., 2008.

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

Leake, D., 1996. *Case-Based Reasoning: Experiences, Lessons and Future Directions*. Menlo Park, California: AAAI Press.

Lee, C.Y., Piramuthu, S. & Tsai, Y.K., 1997. Job shop scheduling with a genetic algorithm and machine learning. *International Journal of Production Research*, 35, p.1171–1191.

Link-Pezet, J., Glize, P. & Gleizes, M.-P., 2000. Abrose: An Adaptive Multi-Agent Tool for Electronic Commerce. *IEEE International Workshops on Enabling Technologies*, p.59.

Lin, F.-C. & Kuo, C.-N., 2002. Cooperative multi-agent negotiation for electronic commerce based on mobile agents. In *IEEE International Conference on Systems, Man and Cybernetics*., 2002.

Lin, G.Y. & Solberg, J.J., 1994. An agent based flexible routing manufacturing control simulation system. In *Proceedings of the 1994 Winter Simulation Conference*., 1994.

Liu, H., Abraham, A., Choi, O. & Moon, S.H., 2006. Variable Neighborhood Particle Swarm Optimization for Multi-objective Flexible Job-Shop Scheduling Problems. In *6th international conference, SEAL 2006*. China, 2006. Springer.

Luck, M., McBurney, P., Shehory, O. & Willmott, S., 2005. Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing).

MacCarthy, B.L. & Jou, P., 1996. Case-based reasoning in scheduling. In MK, K. & CS, W., eds. *Proceedings of the Symposium on Advanced Manufacturing Processes, Systems and Techniques (AMPST96)*., 1996. MEP Publications Ltd.

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

Madureira, A., Fonseca, N. & Pereira, I., 2008b. Self-configuration in Dynamic Scheduling. In *First International Conference on Business Sustainability (BS'08)*. Ofir, Esposende, Portugal, 2008b.

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

Madureira, A., Santos, F. & Pereira, I., 2008a. Self-managing agents for dynamic scheduling in manufacturing. In Keijzer, M., ed. *Proceedings of the 2008 GECCO Conference Companion on Genetic and Evolutionary Computation*. New York, 2008a. ACM.

Madureira, A., Santos, F. & Pereira, I., 2008. Self-managing agents for dynamic scheduling in manufacturing. In Keijzer, M., ed. *Proceedings of the 2008 GECCO Conference Companion on Genetic and Evolutionary Computation*. New York, 2008. ACM.

Madureira, A., Santos, J. & Pereira, I., 2009a. MASDSheGATS – Scheduling System for Dynamic Manufacturing Environments. In S. Ahmed & M.N. Karsity, eds. *MultiAgent Systems*. In-Tech.

Madureira, A., Santos, J. & Pereira, I., 2009b. Hybrid Intelligent System For Distributed Dynamic Scheduling. In *Natural Intelligence for Scheduling, Planning and Packing Problems*. Springer-Verlag.

Maes, P., 1995. Artificial Life meets Entertainment: life like Autonomous Agents. *Communications of the ACM*, 38(11), pp.108-14.

Maniezzo, V., Gambardella, L.M. & De Luigi, F., 2004. Ant Colony Optimization. In G.C. Onwubolu & B.V. Babu, eds. *New Optimization Techniques in Engineering*. Berlin: Springer. pp.101-17.

Mataric, M.J., 1995. Learning in Multi-Robot Systems. In *Adaption and Learning in Multi -Agent Systems, Lecture Notes in Artificial Intelligence*., 1995. Springer.

Maturana, F.P., 1997. *MetaMorph: An Adaptive Multi-Agent Architecture for Advanced Manufacturing Systems*. PhD thesis. The University of Calgary.

Maturana, F., Shen, W. & Norrie, D.H., 1999. MetaMorph: an adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research*, 37(10), pp.2159-73.

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

Metropolis, N. et al., 1953. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21(6), p.1087–1092.

Michalewicz, Z., 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd ed. Berlin: Springer-Verlag.

Minsky, M., 1986. *The Society of Mind*. Simon and Schuster.

Mitchell, T., 1997. *Machine Learning*. McGraw-Hill Education (ISE Editions).

Moscato, P., 1989. *On Evolution, search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms*. C3P Report 826. Caltech Concurrent Computation Program.

Muggleton, S. & Raedt, L.d., 1994. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19, pp.629-79.

Nilsson, N., 1981. *Distributed Artificial Intelligence*. Report SRI International. Menlo Park CA.

Oman, S. & Cunningham, P., 2001. Using case retrieval to seed genetic algorithms. *International Journal of Computational Intelligence and Applications*, 1(1), pp.71-82.

Osman, I. & Kelly, J., 1996. *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers.

Ouelhadj, D., Hanachi, C. & Bouzouia, B., 2000. Multi-agent architecture for distributed monitoring in flexible manufacturing systems (FMS). In *Proceedings of the IEEE International Conference on Robotics and Automation*. San Francisco, USA, 2000.

Ouelhadj, D. & Petrovic, S., 2008. A Survey of Dynamic Scheduling in Manufacturing Systems. *Journal of Scheduling*.

Panait, L. & Luke, S., 2005. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, pp.387-434.

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

Parunak, H.V., 1987. Manufacturing experience with the contract net. In M. Huhns, ed. *Distributed Artificial Intelligence*. Pitman, London. pp.285-310.

Parunak, H.V., 1996. Applications of distributed artificial Intelligence in industry. In G.M.P. O'Hare & N.R. Jennings, eds. *Foundation of Distributed Artificial Intelligence*. New York: Wiley Inter-science.

Pendharkar, P.C., 1999. A computational study on design and performance issues of multi-agent intelligent systems for dynamic scheduling environments. *Expert Systems with Applications*, 16(2), pp.121-33.

Pendrith, M.D., 2000. Distributed Reinforcement Learning for a Traffic Engineering Application. In *Proc. of the Fourth International Conference on Autonomous Agents*. Barcelona, Spain, 2000.

Petrovic, S., Yang, Y. & and Dror, M., 2007. Case-based selection of initialisation heuristics for metaheuristic examination timetabling. *Expert Syst. Appl. 33*, 3 Outubro. pp.772-85.

Plaza, E., Arcos, J.L. & Martin, F., 1996. Cooperative Case-Based Reasoning. In Weiss, G., ed. *Distributed Artificial Intelligence Meets Machine Learning, Lecture Notes in Artificial Intelligence*., 1996. Springer.

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

Ponnambalam, S.G., Aravindan, P. & Rajesh, S.V., 2000. A tabu search algorithm for job shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 6(10), p.765–771.

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

Porter, B. & Bareiss, R., 1986. PROTOS: An experiment in knowledge acquisition for heuristic. In *Proceedings of the First International Meeting on Advances in Learning (IMAL)*. Les Arcs, France, 1986.

Portmann, M.C., 1997. Scheduling Methodology: optimization and compu-search approaches. In *The planning and scheduling of production systems*. Chapman & Hall.

Potter, M., 1997. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis. Fairfax, Virginia: George Mason University.

Potter, M. & Jong, K.D., 2000. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1), pp.1-29.

Quinn, M., 2001. A comparison of approaches to the evolution of homogeneous multi-robot teams. In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001)*. Seoul, Korea, 2001. IEEE Press.

Ramos, C., 1994. An architecture and a negotiation protocol for the dynamic scheduling of manufacturing systems. In *Proceedings of IEEE International Conference on Robotics and Automation*., 1994.

Ramos, C. & Silva, A., 2008. *Agent-based Systems*. Sebenta de Sistemas Baseados em Agentes (SIBAG). Porto: Instituto Superior de Engenharia do Porto.

Ross, B., 1989. Some psychological results on case-based reasoning. In *Case-Based Reasoning Workshop, DARPA 1989*. Pensacola Beach, 1989. Morgan Kaufmann.

Rostami, V. et al., 2005. Cooperative multi agent soccer robot team. In *Proceedings of world academy of science and engineering and technology*., 2005.

Russell, S. & Norvig, P., 1995. *Artificial Intelligence a modern approach*. Prentice-Hall International Editions.

Sandholm, T. & Crites, R.H., 1995. On multiagent Q-learning in a semi-competitive domain. In *Adaption and Learning in Multi-Agent Systems*., 1995.

Schank, R., 1982. *Dynamic memory; a theory of reminding and learning in computers and people*. Cambridge University Press.

Scheutz, M. & Schermerhorn, P., 2003. Many is More, But Not Too Many: Dimensions of Cooperation of Agents with and without Predictive Capabilities. In *Intelligent Agent Technology, IEEE / WIC / ACM International Conference*., 2003.

Schirmer, A., 2000. Case-based reasoning and improved adaptive search for project scheduling. *Naval Research Logistics 47*, pp.201-22.

Schmidhuber, J., 1996. Realistic multi-agent reinforcement learning. In *Learning in Distributed Artificial Intelligence Systems, Working Notes of the 1996 ECAI Workshop*., 1996.

Schmidhuber, J. & Zhao, J., 1996. Multi-agent learning with the success-story algorithm. In *ECAI Workshop LDAIS/ ICMAS Workshop LIOME*., 1996.

Schmidt, G., 1998. Case-based reasoning for production scheduling. *International Journal of Production Economics*, 56-57, pp.537-46.

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

Shaw, J.M., 1988. Dynamic scheduling in cellular manufacturing systems: a framework for Network decision making. *Journal of Manufacturing Systems*, 7(2), pp.83-94.

Shen, W., Maturana, F. & Norrie, D.H., 1998. Learning in Agent-Based Manufacturing Systems. *Proceedings of AI & Manufacturing Research Planning Workshop*, pp.177-83.

Shen, W., Maturana, F. & Norrie, D.H., 2000. Metaphor II: an agent-based architecture for distributed intelligent design and manufacturing. *Journal of Intelligent Manufacturing*, 11(3), pp.237-51.

Shen, W. & 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), pp.129-56.

Shen, W., Norrie, D.H. & Barthes, J.P.A., 2001. *Multi-agent systems for concurrent intelligent design and manufacturing*. London: Taylor & Francis.

Shi, B. & Huang, W., 2008. Modeling and Development of Multi-agent Traffic Control Experimental System Based on Petri Net. In *International Conference on Intelligent Computation Technology and Automation*., 2008.

Smith, R., 1980. The contract net protocol: high level communication and control in distributed problem solver. *IEEE Transactions on Computers*, 29(12), pp.1104-13.

Smith, D., Cypher, A. & Spohrer, J., 1994. KidSim: programming agents without a programming language. *Communications of the ACM*, 37(7), pp.54-67.

Spall, J.C., 2003. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc.

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

Stone, P., 2000. *Layered Learning in Multiagent system: A Winning Approach to Robotics Soccer*. MIT Press.

Stone, P. & Veloso, M., 1997. Towards collaborative and adversarial learning: A case study for robotic soccer. *International Journal of Human Computer Studies*, 48.

Sule, D., 1997. *Industrial Scheduling*. PWS Publishing.

Sun, J. & Xue, D., 2001. A dynamic reactive scheduling mechanism for responding to changes of production orders and manufacturing resources. *Computers in Industry*, 46(2), pp.189-207.

Sutton, R., 1988. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, pp.9-44.

Sutton, R.S. & Barto, A.G., 1998. *Reinforcement Learning*. The MIT Press.

Sweitzer, J. & Draper, C., 2006. Architecture Overview for Autonomic Computing. In *Autonomic Computing – Concepts, Infrastructure, and Applications*.

Tan, M., 1993. Multi -Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proceedings of the Tenth International Conference on Machine Learning*., 1993.

Toriz, A., Sánchez, A. & Osorio, M., 2009. Coordinated multi-agent exploration. *Journal Research in Computing Science*, 42, pp.189-200.

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

Viamonte, M.J., Ramos, C., Rodrigues, F. & Cardoso, J., 2006. ISEM: A Multi-Agent Simulator For Testing Agent Market Strategies. In *IEEE Transactions on Systems, Man and Cybernetics*., 2006.

Watkins, C. & Dayan, P., 1992. Q-learning. *Machine Learning*, 8, pp.279-92.

Weinberg, M. & Rosenschein, J., 2004. Best-response multiagent learning in non-stationary environments. In *AAMAS-2004 — Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*., 2004.

Wellner, J. & Dilger, W., 1999. Job shop scheduling with multiagents. In *Workshop Planen und Konfigurieren*., 1999.

Williams, A.B., 2004. Learning to Share Meaning in a Multi-Agent System. *Autonomous Agents and Multi-Agent Systems*, 8(2), pp.165-93.

Wooldridge, M. & Jennings, N., 1995. Agent Theories, Architectures, and Languages: a Survey. In *Intelligent Agents*. Springer-Verlag. pp.1-22.

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

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

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

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

Zambonelli, F. & Parunak, H., 2004. Towards a Paradigm Change in Computer Science and Software Engineering: A Synthesis. *The Knowledge Engineering Review*, 18(4).

Zhang, H.-T., Yu, F. & Li, W., 2009. Step-coordination algorithm of traffic control based on multi-agent system. *International Journal of Automation and Computing*, 6(3), pp.308-13.

Zwaan, S.v.d., Pais, A.R. & Marques, C., 1999. *Ant Colony Optimisation for Job Shop Scheduling*.