INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Aplicação de Métodos Emergentes em Problemas de Escalonamento do Tipo *Flexible Job Shop*

Inês Cristina Vinhas de Seixas

Relatório Final da Dissertação apresentado à

Escola Superior de Tecnologia e Gestão

Instituto Politécnico de Bragança

para obtenção do grau de Mestre em

Engenharia Industrial

Ramo de especialização em Engenharia Eletrotécnica

Orientador:
Prof. Dr. Paulo Leitão
Co-orientador:
Prof. Dra. Ana Isabel Pereira

Novembro, 2013

Dedicatória

Ao José, à Mariana e ao Gonçalo.

Agradecimentos

Primeiro de tudo, quero expressar o meu agradecimento especial ao Professor Paulo Leitão e à Professora Ana Isabel pelo seu conhecimento, paciência, apoio e motivação constante durante o desenvolvimento deste trabalho, sem a qual a conclusão não seria possível.

Em seguida, eu quero agradecer a todos os meus amigos pelo apoio, incentivo e motivação.

Também quero agradecer aos meus pais por todo o apoio e amor ao longo dos anos.

Por último, mas não menos importante, quero agradecer ao meu marido José, pelo seu apoio e compreensão e aos meus filhos Mariana e Gonçalo por me fazerem sorrir constantemente.

Resumo

Hoje em dia, os sistemas de fabrico assumem um papel importante para atingir níveis de competitividade adequados aos desafios emergentes com que as empresas são confrontadas no atual mercado globalizado. Estes sistemas são tradicionalmente complexos, quer em dimensão quer em inovação técnica. A acrescentar a estes fatores está o crescente requisito de demanda de produtos altamente customizados e de elevada qualidade por parte dos clientes, aliado a ciclos de vida cada vez mais curtos dos produtos.

Estes requisitos, entre outros, fazem com que técnicas inovadoras de escalonamento sejam necessárias para que de uma forma rápida e eficaz, os gestores destes sistemas possam, distribuir e "orquestrar" de forma eficiente os trabalhos necessários a realizar no sistema produtivo, de forma a maximizar o lucro da empresa.

O presente trabalho estuda diversas técnicas de escalonamento existentes, nomeadamente técnicas baseadas em meta-heurísticas, e que podem ser utilizados em sistemas de fabrico flexível. De forma particular o método baseado em algoritmos genéticos é detalhado e aplicado a um caso de estudo real de uma célula de fabrico flexível existente na Université de Valenciennes et du Hainaut-Cambrésis.

Palavras-chave: escalonamento, meta-heurísticas, algoritmos genéticos, sistemas de manufatura flexível.

Abstract

Nowadays, manufacturing systems have a crucial role in order to achieve suitable competiveness levels to the emergent challenges that companies are facing daily. These systems are usually complex in terms of dimension and technical innovation. Additionally to these constraints is the request for highly customized products with higher quality by costumers, allied to even shorter product life-cycle.

These requirements, among others, make that scheduling techniques are necessary for, in fast and reliable way, that system managers can, distribute and orchestrate in an effective way, the necessary works to be made in the productive system, in order to maximize the companies' profit.

The present work studies several scheduling techniques, namely those based in meta-heuristics, and which can be used in flexible manufacturing systems. In a particular way, the genetic algorithm method is detailed and is applied into a real case study of a flexible manufacturing cell located at the Université de Valenciennes et du Hainaut-Cambrésis.

Keywords: scheduling, meta-heuristics, genetic algorithms, flexible manufacturing systems

Acrónimos

ACO	*Ant Colony Optimization*
AG	Algoritmo Genético
FIFO	*First In First Out*
FMS	*Flexible Manufacturing System*
FSJ	*Flexible Job Shop*
FJSP	*Flexible Job Shop Problem*
FJSS	*Flexible Job Shop Scheduling*
JS	*Job Shop*
JSP	*Job Shop Problem*
JSS	*Job Shop Scheduling*
JSSP	*Job Shop Scheduling Problem*
MILP	Programação Linear Inteira Mista
MILNP	Programação Não-Linear Inteira Mista
MIP	Programação Inteira Mista
PFSP	*Permutation Flow Shop Scheduling Problem*
PL	Programação Linear
PNL	Programação Não Linear
PSO	*Particle Swarm Optimization*
PWA	*Pice Wise Affine*
SA	*Simulated Annealing*
SI	*Swarm Intelligence*
TS	*Tabu Search*
TSP	*Traveling Salesman Problem*

Índice

Lista de figuras

Lista de Tabelas

Capítulo 1

Introdução

Diariamente somos deparados com a obrigatoriedade do uso de produtos, podendo eles serem considerados bens físicos, serviços, pessoas, organizações, ou simples planos. Segundo *Kotler* e *Armstrong* [1], "*produto é qualquer coisa que possa ser oferecida a um mercado para atenção, aquisição, uso ou consumo, e que possa satisfazer a um desejo ou necessidade*".

1.1 Contextualização

Todos os produtos podem ser transformados em novos produtos dependendo das nossas necessidades, e da exigência ao longo dos tempos, havendo assim uma análise constante aos produtos fabricados. Todo este processo é designado por ciclo de vida do produto, que compreende quatro fases muito importantes: introdução, crescimento, maturação e declínio.

Na fase de introdução do bem, ou produto, há uma venda inicial baixa que cresce lentamente, sendo esta fase mais arriscada e cara, pelo fato dos produtos não serem imediatamente aceites pelo mercado. Nesta fase são utilizadas diversas técnicas de suporte, nomeadamente técnicas de previsão e a pesquisa direta com o consumidor, questionando-o sobre a necessidade de um dado produto.

Na fase de crescimento, as vendas e o lucro crescem rapidamente, pois o mercado encontra-se em expansão e começa a aceitar o produto.

Numa terceira fase, as vendas passam a ter um crescimento mais lento devido à saturação do produto e ao aparecimento de produtos concorrentes com novas funcionalidades e/ou novo *design*. Nesta fase, as companhias complementam a sua oferta ao fornecer serviços complementares e também começam a introduzir novos produtos.

Verificando-se que os lucros diminuem de dia para dia, entramos numa fase de declínio, que pode ter ocorrido pela introdução de produtos mais eficazes; a substituição de um produto por outro melhor, ou ainda pela sua obsolescência.

Hoje em dia, a produção é entendida como o processo de conversão de matéria-prima num produto físico. Sendo assim, um sistema de fabrico refere-se a qualquer conjunto de elementos que convertem matéria-prima em produto. Esses produtos podem no entanto ser usados para a produção de outros produtos mais complexos. No entanto, é necessário que os produtos satisfaçam os clientes, nas datas e prazos estipulados, com a qualidade exigida e sob os princípios de racionalização de minimização de custos e tempo de produção. A Figura 1 apresenta uma breve esquematização sobre os vários componentes existentes numa empresa e que são necessários para o desenvolvimento de um produto.

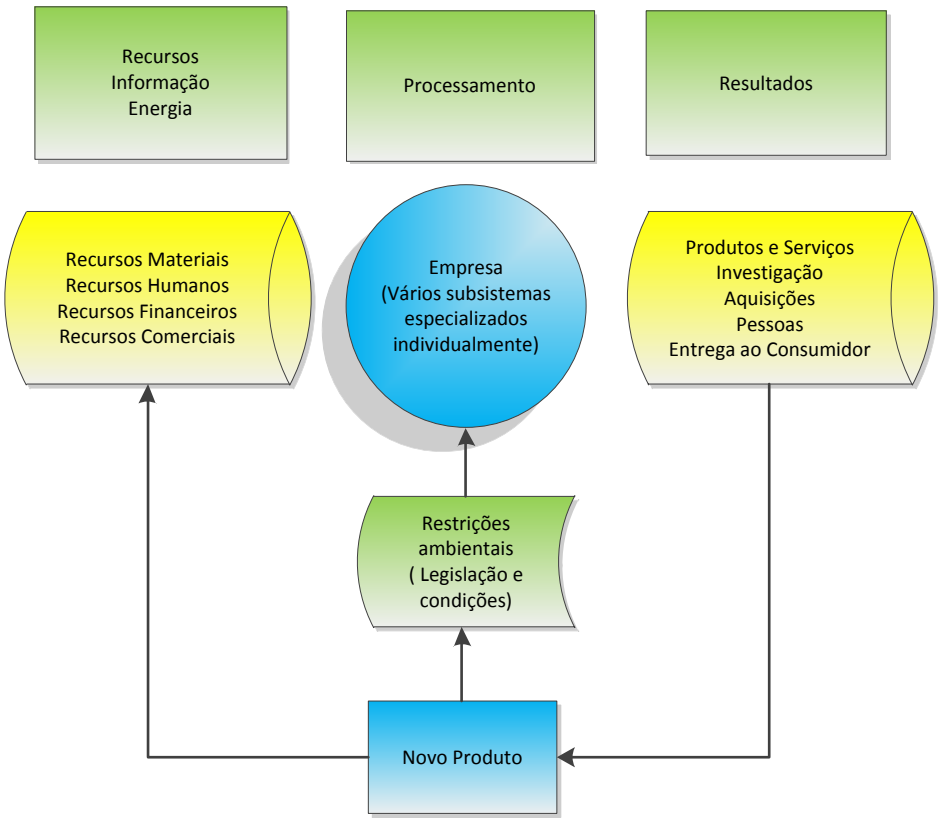

Figura 1- Funcionamento de um sistema de fabrico.

Num sistema de fabrico são produzidos produtos, que podem variar em termos de escala de produção, e pretende-se que sejam vendidos com uma margem de lucro.

Os sistemas de fabrico, tendo em conta as quantidades produzidas podem ser classificados, segundo *Groover* [2] de três formas:

- *Produção em oficina*, conhecida também por produção *Job Shop*, onde são produzidas grandes variedades de produtos, sendo que a produção de cada um é realizada em pequena quantidade;

- *Produção em massa*, onde ocorre o extremo da produção em oficina, isto é, a produção é feita em grandes quantidades para cada produto mas com pouca variedade de produtos.
- *Produção em lotes*, que é definida como o meio-termo entre a produção em oficina e em massa, caracterizando-se pela produção de média variedade em média quantidade.

A evolução da tecnologia e a diminuição do poder de compra fez com que houvesse uma diminuição das quantidades a produzir de cada produto e o aumento da variedade de produtos a serem requeridos pelo mercado. Uma produção que no passado era feita em massa, hoje em dia é feita em lotes.

Em termos de *layout*, ou seja, a forma como os equipamentos, espaços para armazenamento, corredores de circulação, entre outros, estão distribuídos pelo espaço da fábrica, é possível encontrar três tipos de produção [2]:

- *Layout* de posição fixa, no qual o produto, devido às suas grandes dimensões, está fixo e são as máquinas e operários que se deslocam para realizarem as operações. A construção de barcos ou casas são exemplos típicos deste tipo de *layout*.
- *Layout* de fluxo de produto, no qual as estações de trabalho estão organizadas de forma a minimizar o transporte entre elas, tendo em consideração o plano de processo do produto. Exemplos deste tipo de produção são as linhas de montagem na indústria automóvel.
- *Layout* de processo, no qual as máquinas estão organizadas consoante o tipo de processo que executam, permitindo uma maior flexibilidade na produção de produtos com alguma variedade. Neste tipo de *layout*, é possível verificar uma agregação de estações de soldadura, estações de pintura, de centros de torneamento, e estações de inspeção.

Assim, um sistema de fabrico consiste numa estrutura organizada de recursos físicos necessários à execução das funções de fabrico, sendo vulgarmente utilizados sistemas de automação com tecnologia avançada, como sejam robôs, máquinas de controlo numérico, tapetes automáticos e sistemas de armazenamento automático, tal como é ilustrado no exemplo da Figura 2.

Figura 2- Sistema automatizado [3].

No entanto, um sistema de fabrico será de pouca utilidade sem a presença de um sistema de controlo apropriado, que seja responsável pela execução física dos planos de produção, organizando, sincronizando e monitorizando o progresso do produto que está a ser processado, montado, transportado ou inspecionado na fábrica.

O sistema de controlo de fabrico compreende vários componentes, tal como é ilustrado na Figura 3: planeamento, escalonamento, despacho, monitorização, diagnóstico e recuperação de erros.

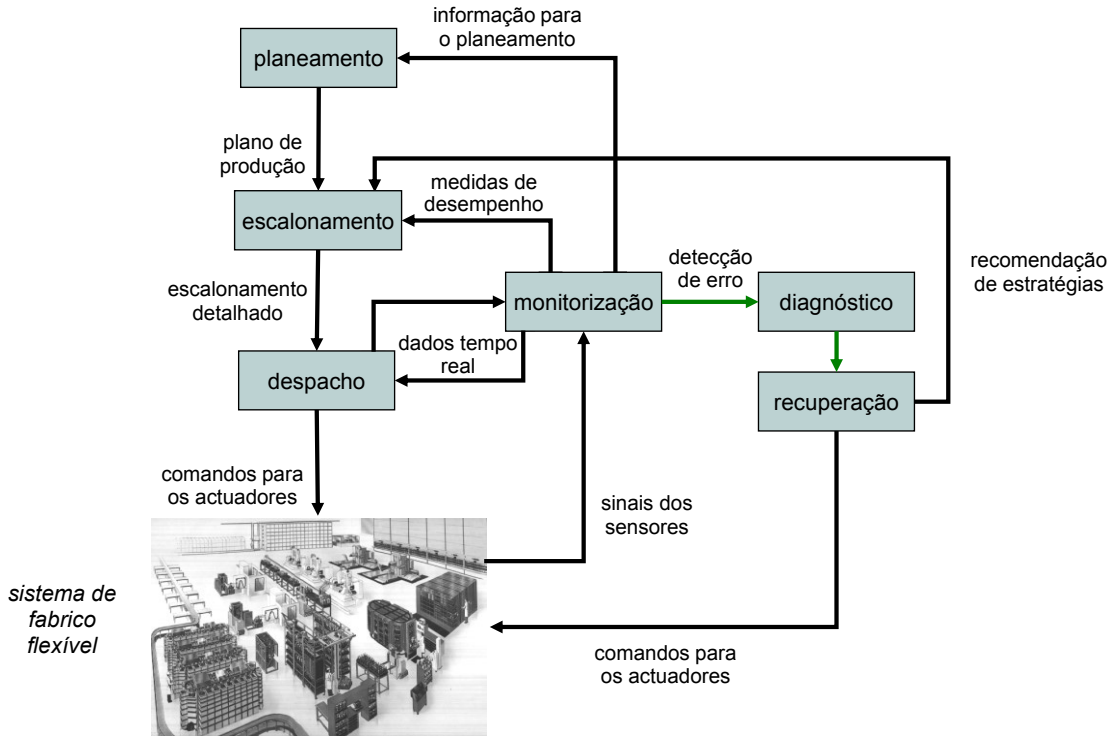

Figura 3- Arquitetura típica de um sistema de controlo de fabrico [4].

Em particular, o escalonamento é referente à alocação ótima de recursos a tarefas ao longo do tempo, obedecendo a um conjunto de restrições que refletem as relações temporais entre tarefas e a capacidade limitada dos recursos. Em particular este algoritmo deve decidir i) em que altura se deve produzir os produtos encomendados, ii) que quantidade de cada produto se deve produzir e iii) como e quando usar os recursos para produzir os produtos.

O problema de escalonamento é tipicamente um problema combinatório complexo, mais propriamente um problema não polinomial NP (por exemplo para um problema de n tarefas e m máquinas, o número de soluções é dado por $(n!)^m$). Adicionalmente, o problema de escalonamento torna-se ainda mais complexo uma vez que os sistemas de fabrico são sujeitos a perturbações, e.g. atrasos e avarias nas máquinas, e são sistemas não-lineares, complexos e caóticos.

1.2 Motivação

A motivação deste trabalho é compreender as várias técnicas que podem ser usadas para resolver problemas complexos de engenharia, nomeadamente no domínio dos sistemas de fabrico do tipo *Job Shop*, e estudar a aplicação de técnicas emergentes neste domínio.

Para este fim, serão estudados vários algoritmos de escalonamento e algumas aplicações existentes, sendo analisada a utilização de diferentes métodos para diferentes domínios de aplicação.

Uma das técnicas estudadas com maior detalhe foi a de algoritmos genéticos, inspirada nos princípios da teoria da evolução introduzida por *Charles Darwin*, e na qual os problemas são resolvidos através de um processo evolutivo que conduz à melhor solução. Na perspetiva de demonstrar o elevado grau de eficiência na técnica de algoritmos genéticos para obter soluções ótimas na otimização deste tipo de problemas, foi utilizado um caso de estudo real, onde se pretende minimizar o tempo de produção.

O caso de estudo utilizado é a célula de fabrico flexível AIP-PRIMECA, instalada na Université de Valenciennes et du Hainaut-Cambrésis em França, que compreende sete estações de trabalho interligadas por um sistema de tapetes automatizado e permite executar um conjunto diversificado de produtos.

1.3 Organização do relatório

Este documento está organizado em seis capítulos, começando com o presente capítulo, onde a contextualização do problema e os objetivos foram apresentados.

O segundo capítulo, intitulado "Problemas de Escalonamento em Sistemas de Fabrico", apresenta uma visão geral sobre o problema de escalonamento em sistemas de fabrico, sendo caraterizados os vários tipos de escalonamento usualmente encontrados neste domínio. O terceiro capítulo, com o título "Métodos de Otimização", examina a aplicação de diversas técnicas de otimização para resolver este tipo de problemas de escalonamento e, em particular problemas complexos encontrados em *Job Shop*. O quarto capítulo, intitulado "Caso de Estudo", descreve pormenorizadamente o caso de estudo real onde vai ser implementado um dos métodos de otimização descrito no capítulo anterior, neste caso o método baseado em algoritmos genéticos. O capítulo 5 descreve a implementação do modelo matemático descrito no capítulo anterior, usando a abordagem de algoritmos genéticos, e analisa os resultados obtidos. O capítulo 6, intitulado "Conclusões e Trabalho Futuro", apresenta as conclusões do trabalho realizado e aponta algumas ideias de trabalho futuro.

Capítulo 2

Problemas de Escalonamento em Sistemas de Fabrico

O escalonamento é uma ferramenta importante na engenharia, e em particular na indústria, que promove a competitividade das empresas, sendo crucial a aplicação de algoritmos eficientes que permitam obter os produtos de uma forma mais rápida e eficaz, e que pode ter um impacto importante sobre a produtividade de um processo. Sem a existência de escalonamento de tarefas não é possível obter eficiência, competitividade e rentabilidade nas empresas.

2.1 Sistemas Fabrico Flexíveis

Nos sistemas de manufatura, o conceito de flexibilidade aparece como um elemento essencial para serem geridos. A flexibilidade de um sistema de produção, é sua capacidade de adaptação a um grande número de mudanças [5-6].

Com respeito às incertezas ambientais, a flexibilidade na manufatura é necessária para lidar com mudanças internas e forças externas. Os distúrbios internos incluem a quebra de equipamentos, variações nos tempos das tarefas e esperas em filas. Forças externas referem-se usualmente às incertezas fundamentais do ambiente de competição. Tais incertezas podem estar presentes na disponibilidade de recursos, na variedade e preço dos produtos. Além disso, podem significar mudanças nas escolhas de consumidores, inovações tecnológicas, novas regulamentações, etc [7].

Um sistema de manufatura flexível (FMS) é um tipo de processo industrial que permite que os equipamentos sejam utilizados para mais de uma finalidade, sendo relacionados de alguma forma. Desta forma são gerados vários problemas de escalonamento complexos, sendo necessário o uso de algoritmos eficientes para os resolver.

2.2 Definição de escalonamento

O escalonamento define onde e quando vão ser realizadas as tarefas relativas à produção dos produtos, de forma a maximizar o lucro de um dado produto. Assim, diversas definições, mas nunca contraditórias, podem ser encontradas na literatura. Entre elas, pode-se encontrar uma que define escalonamento como "*o trabalho ou atividade de escalonar os horários em que determinadas tarefas serão realizadas ou eventos acontecer*" [8] ou uma outra em que o define como "*determinar quando uma atividade deve começar ou acabar, dependendo da sua duração, da(s) atividade(s) precedente(s), das relações precedentes, da disponibilidade dos recursos e do objetivo de conclusão do projeto*" [9].

De salientar que a otimização deste tipo de problemas está classificada como sendo do tipo NP-*hard*. O exemplo mais simples, e mais usado na literatura, é o caixeiro-viajante. O problema é muito simples de descrever: imagine-se que um caixeiro-viajante tem que visitar *n* cidades e que só as pode visitar uma só vez e que, naturalmente, deseja fazer o caminho mais curto e que no final do percurso regressar a casa. Este problema muito simples, torna-se num problema de escalonamento muito complexo se notarmos que o número de possibilidades de caminhos é de *n!*. Note-se que, por exemplo, para somente 15 cidades, o número de possibilidades de caminhos será de 1.3077×10^{12}.

Por outro lado, os algoritmos de escalonamento visam otimizar parâmetros relacionados com a produção, por exemplo, maximizar o tempo de utilização dos recursos ou minimizar o tempo de transporte. Alguns dos critérios de avaliação de desempenho, dependendo do que se pretende, podem ser:

- *Makespan* - tempo total de processamento de todos os trabalhos (mais comum)
- Ciclo de produção médio;
- Ciclo de produção médio ponderado;
- Atraso algébrico máximo;
- Atraso médio ponderado;
- Número de produtos em atraso.

No presente estudo, o critério de otimização a utilizar é dos mais usados, *makespan*, e prende-se com a minimização do tempo que vai desde que uma ordem chega ao sistema, por exemplo, uma ordem de um lote de produtos, até ao tempo em que a ordem fica completamente processada.

Para visualização do resultado do escalonamento, para situações menos complexas, utiliza-se uma técnica simples de representação gráfica (Gráficos de Gantt) que não prevê quaisquer regras para a escolha, mas simplesmente apresenta visualização de resultados, plano de processamento e avaliação do *makespan*, do tempo de espera, do tempo de processamento, da utilização da máquina, etc [10].

Um exemplo de um escalonamento representado num diagrama de Gantt é mostrado na Figura 4, onde se pode ver a distribuição das operações pelos recursos existentes.

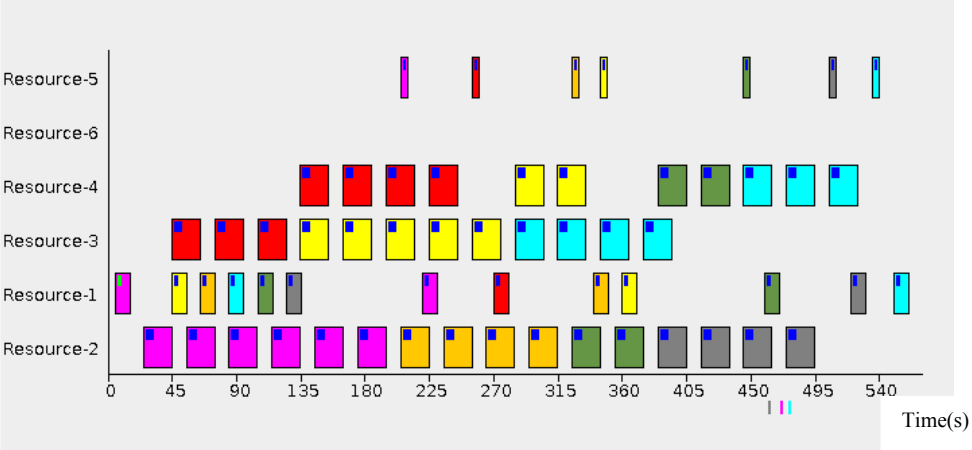

Figura 4 – Exemplo de um diagrama de Gantt.

No desenvolvimento de uma solução de escalonamento, é necessário implementar dois passos importantes:

a) Definição do problema, descrevendo com exatidão o que se pretende, nomeadamente definir o modelo matemático que defina as variáveis a controlar, a função objetivo e o conjunto de restrições.
b) Seleção do algoritmo de escalonamento mais conveniente e eficaz conforme o objetivo final que se pretende atingir.

O escalonamento obriga a que haja uma orientação no caminho a seguir para um projeto ser executado e define metas e objetivos a serem alcançados em tempo útil para a sua conclusão. A escolha do tipo de escalonamento que se vai utilizar depende essencialmente do que se pretende obter, i.e. da função objetivo.

Diversos critérios podem ser aplicados de forma a avaliar a qualidade do serviço oferecido por um algoritmo de escalonamento, como seja a qualidade da solução obtida (dependente do grau de otimização da função objetivo) e o tempo de resposta (i.e. tempo necessário para determinar a solução de escalonamento).

Alguns dos benefícios da utilização de escalonamento num sistema de fabrico são a redução de stocks, redução do esforço de programação, aumento da eficiência da produção, nivelamento de carga de trabalho e data de entrega mais precisa.

2.3 Escalonamento em *Job Shop*

A configuração do sistema produtivo tem uma forte ligação com o tipo de problemas de escalonamento e consequentemente no tipo de técnicas de escalonamento a serem utilizadas. Nesta secção serão analisados os diferentes problemas de escalonamento usualmente encontrados em sistemas de fabrico.

O *Job Shop Scheduling* (JSS), ou *Job-Shop Problem* (JSP), surge na década de 50, é um problema de otimização que aloca várias operações a determinados recursos. Este tipo de problema procura encontrar uma sequência de operações para cada máquina, respeitando as restrições impostas, otimizando um determinado objetivo [11].

Assim, o objetivo de um problema JSP é produzir uma sequência ordenada de operações para ser processada em cada máquina. O tempo de início de cada operação é atribuído de acordo com a referida sequência, de forma a otimizar uma função objetivo desejada.

A forma mais básica em que podemos traduzir o problema JSP é através de um simples exemplo em que existem *n* trabalhos, cada um dos quais compostos por várias operações que têm que ser realizados em *m* máquinas. Neste problema é necessário considerar algumas restrições:

- Cada operação *Oij* apenas é escalonada uma vez.
- Cada operação *Oij* pode ser escalonada para qualquer máquina de um dado conjunto de máquinas.
- Cada máquina apenas pode processar uma operação de cada vez (por um período de tempo T_{ij}, necessário ao seu processamento).
- Nenhuma máquina pode ser libertada antes de finalizada a execução da operação (i.e. não existe a possibilidade de interrupção na execução das operações).
- O número total de máquinas de cada tipo é fixo.

O escalonamento JSP pode ser determinado através de alguns fatores, tais como, a ordem de chegada, o número de máquinas, a sequência de trabalho e o critério de avaliação de desempenho.

Relativamente à ordem de chegada das tarefas, existem duas abordagens distintas:

- Estática: *n* tarefas chegam em simultâneo e é necessário agendar o trabalho;
- Dinâmica: a chegada é realizada de forma intermitente (muitas vezes estocástica, ou seja, de forma aleatória);

Em relação à sequência de trabalho também temos duas formas distintas de processamento, tal como se apresenta de seguida.

2.3.1 Escalonamento em *Flexible Job Shop*

O *Flexible Job Shop scheduling* (FJSS) ou *Flexible Job Shop problem* (FJSP) é uma das variantes do JSP que permite que uma determinada operação seja realizada a partir de qualquer máquina de um determinado conjunto de máquinas. O problema encontra-se na atribuição de cada operação a cada máquina e a sua ordenação, de forma que o tempo máximo de duração para conclusão de uma operação (*makespan*) seja minimizado.

O FJSP pode ser descrito por *n* operações para serem processados em *m* máquinas. Cada operação necessita ser executada, e cada uma destas deve ser executada num tempo fixo numa determinada máquina. Existem várias restrições que devem ser consideradas, nomeadamente:

- Cada trabalho é constituído por um conjunto de operações.
- Cada operação só poderá passar numa máquina uma única vez.
- Têm que ser respeitadas as diferentes precedências de operações que poderão ocorrer.
- Cada máquina só pode processar uma operação de cada vez.

Podemos então resumir o FJSS da seguinte forma, é dado um conjunto $J = \{J_1, J_2, \ldots, Jo\}$ de trabalhos independentes. Cada J_i é constituído por uma sequência de operações O_{i1}, O_{i2}, , O_{in} de operações a serem realizadas uma após a outra de acordo com a sequência indicada [12].

É dado um conjunto $U = \{M_1, M_2, \ldots, M_m\}$ de máquinas. Cada operação O_{ij} pode ser executada por qualquer máquina. O tempo de processamento de cada operação é dependente da máquina. Cada operação deve ser concluída sem interrupção uma vez iniciada. Além disso, as máquinas não podem desempenhar mais do que uma operação de cada vez. Todos os trabalhos e máquinas estão disponíveis no momento inicial [13].

Existe uma grande variedade de problemas do mundo real que podem ser modelados como um FJSP, por exemplo, para otimizar operações em sistemas de simulação, otimização em sistemas de transporte e em sistemas de produção.

2.3.2 Escalonamento em *Flow Job Shop*

No problema de escalonamento *Flow Shop Scheduling Problem* (FSSP), o controlo do fluxo do processo deve permitir uma sequência apropriada para cada trabalho e para um conjunto de máquinas, em conformidade com as ordens de processamento. Especialmente, deve ser mantido um fluxo contínuo de tarefas de processamento com um tempo mínimo de inatividade e um mínimo de tempo de espera. Este tipo de escalonamento pode ser encontrado, classicamente, em sistemas de fabrico.

Um tipo especial de FSSP é a *Permutation Flow Shop Scheduling Problem* (PFSSP), em que a ordem de processamento dos trabalhos sobre os recursos é o mesmo para cada etapa subsequente de processamento.

Os primeiros valores de referência, i.e. os primeiros *lower bounds*, de JSSP foram apresentados por *Fisher* e *Thomson* em 1963 [14]. Desde essa altura, este tipo de problemas despertou a atenção da comunidade e um grande número de investigadores têm estudado este problema e proposto diversos métodos de resolução, nomeadamente métodos exatos e de aproximação. Métodos exatos, como programação linear e convergência Lagrangeana têm sido bem sucedidos na resolução de pequenos problemas, embora seja necessária uma programação mais complexa uma vez que o tamanho do problema aumenta. Dado isto, os investigadores desviaram a sua atenção para a meta-heurística e inteligente *Hybrid Search* para resolver JSSP. Entre estes, o *shifting bottleneck approach*, *particle swarm optimization*, *ant colony optimization*, *simulated annealing*, *tabu search*, *genetic algorithm*, *neural network* e *immune algorithm* são os exemplos mais típicos. Esses algoritmos de otimização inteligentes são relativamente fáceis de implementar e podem ser convenientemente adaptados a diferentes tipos de problemas de programação e as suas soluções de alta qualidade podem ser encontradas com um tempo de programação razoável [15].

Na secção seguinte faz-se uma descrição mais detalhada de alguns métodos de otimização que podem ser usados para a resolução deste tipo de problemas.

Capítulo 3

Métodos de Otimização

Os algoritmos de otimização podem ser classificados em quatro classes: os métodos determinísticos, heurísticos, ponto a ponto e populacionais.

- **Métodos Determinísticos**: São métodos que usam modelos matemáticos bem definidos de forma a identificar o ótimo do problema a minimizar. Dependendo da complexidade do problema tratado, este tipo de métodos pode consumir muito tempo para obter a solução ótima [16].
- **Métodos Heurísticos**: Os métodos são baseados num conjunto de regras, que podem gerar padrões aleatórios, de forma a obter soluções, próximas da solução ótima, sem, no entanto, garantir a solução ótima. Os modelos heurísticos são caracterizados pela obtenção de boas soluções para um problema em tempos de processamento viáveis [17].
- **Métodos Ponto-a-Ponto**: A procura pela melhor solução processa-se sempre de um único ponto para outro (inicia-se com um único candidato), no espaço de decisão, através da aplicação de alguma regra de transição, ou seja, o método realiza a procura sempre na vizinhança do ponto corrente, constituindo, portanto, um método de procura local [18].
- **Métodos Populacionais**: São métodos que, em cada iteração, possuem diversas aproximações à solução. Em geral, estes métodos necessitam de um elevado número de avaliações à função para convergir para a solução do problema.

Na sua forma geral, nos problemas de otimização o importante é conseguir maximizar ou minimizar uma função num domínio restrito. A modelação matemática faz parte integrante da resolução, ou seja, deve ser construída uma representação matemática do problema para que se perceba a sua essência.

Diariamente deparamo-nos com problemas que de alguma forma podem ser vistos como problemas de otimização, tais como, o percurso mais rápido entre cidades, a melhor forma de organizar objetos, entre outros. Resolver tais problemas pode ser entendido como a procura da melhor solução num espaço de potenciais soluções.

Reduzir o tempo de caminho, poupar para comprar algum objeto, tomar decisões, são algumas questões que se colocam quando queremos maneiras ótimas de aplicar os nossos recursos. Resolver um problema de otimização, significa sobretudo procurar a solução de um problema de forma a maximizar ou a minimizar algo. Esse algo tem uma representação matemática que recebe o nome de função objetivo.

A procura de uma solução mais adequada entre diversas soluções alternativas remete-nos para um problema de otimização. Neste capítulo vão ser abordados os vários métodos de otimização para resolução de problemas nos sistemas industriais.

Um problema de otimização pode ser colocado na seguinte forma geral,

$$\min f(x)$$

$$\text{s. a } g(x)<=0$$

Dependendo do comportamento da função objetivo *f(x)* e de como o conjunto é descrito, temos diferentes classes de problemas de otimização, para os quais uma variedade de métodos de solução tem sido desenvolvida.

Na Figura 5 resumem-se as principais técnicas utilizadas [19] para resolver problemas de escalonamento em *job shop* e cada um define que método melhor se enquadra para resolução, i.e., perceber se estamos perante um problema de aproximação ou de otimização. Além disso é necessário saber se se trata de um processo iterativo ou não. Caso o seja, a sua solução é obtida a partir dos dados do problema, se não for a sua solução é procurada continuamente de forma iterativa.

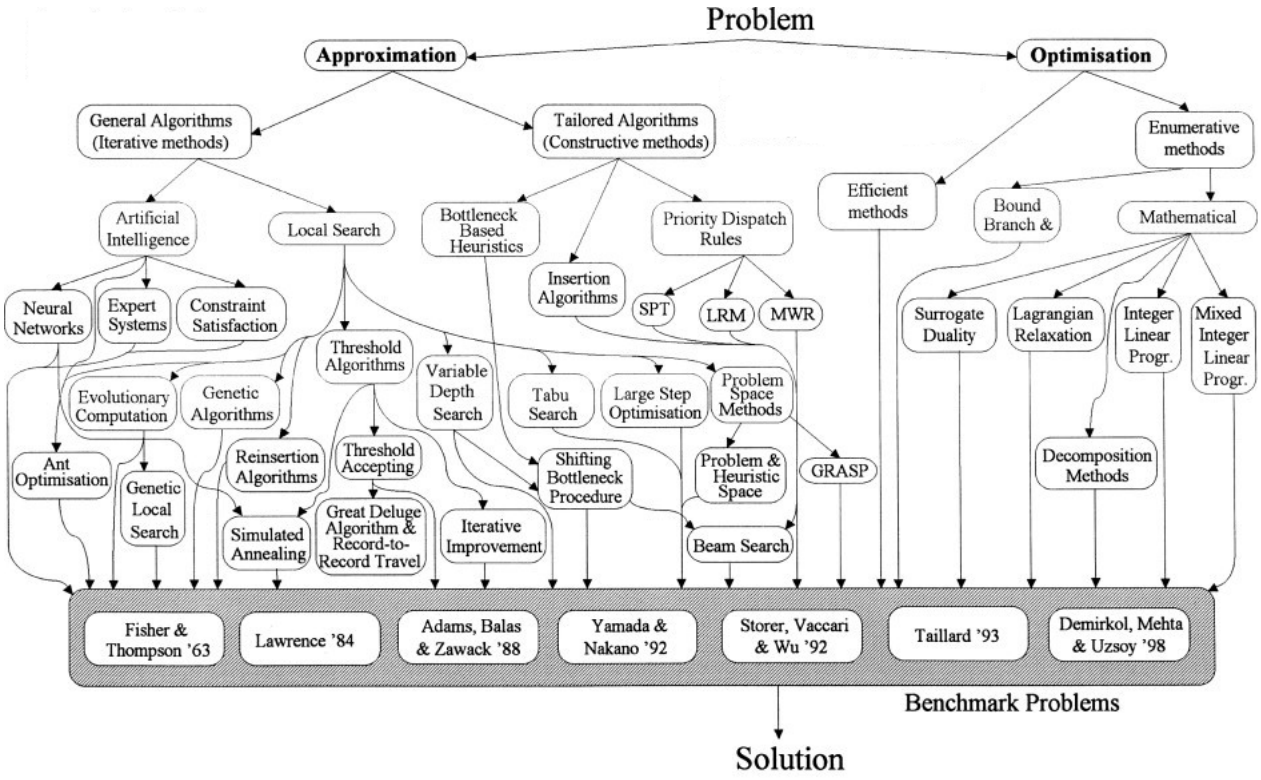

Figura 5 – Classificação de métodos de otimização em *job shop* [19].

Os métodos por aproximação, como sejam os baseados em inteligência artificial, e.g. redes neuronais, e os baseados em pesquisas locais, e.g. algoritmos genéricos, apresentam uma elevada rapidez na resolução do problema de escalonamento mas nem sempre é encontrada uma solução ótima, enquanto nos métodos por otimização, como por exemplo os baseados em métodos enumerativos, e.g. programação linear, a solução global é mais próxima da solução ótima embora o tempo de processamento seja mais longo e a necessidade de memória é maior.

3.1 Programação Linear

Os problemas de Programação Linear (PL) são problemas de otimização nos quais a função objetivo e as funções restrição são todas lineares, ou seja, são equações matemáticas em que nenhuma variável independente é elevada a uma potência maior que um [20].

A programação linear inteira mista é um quadro muito geral para problemas onde as variáveis variam em espaços contínuos e discretos. Alguns exemplos de problemas lineares inteiros mistos são:

- Problemas de atribuição.
- Controlo de sistemas híbridos.
- Sistemas *Piecewise-affine* (PWA) (incluindo aproximações de sistemas não-lineares).
- Problemas com caracteres não-convexos restrições (por exemplo, anti-colisão).

Os métodos mais usados para a resolução de problemas do tipo PL, particularmente os problemas linear inteiro misto (MILP), que são os que precisam que algumas ou todas as variáveis sejam do tipo inteiro, são os métodos de pontos interiores ou *Simplex*.

3.1.1 Método *Simplex*

O método *Simplex* [21] é uma técnica utilizada para determinar, numericamente, a solução ótima de um modelo de PL. É muito utilizado pois é facilmente implementado e consegue resolver problemas com muitas variáveis. Produz variáveis auxiliares para análise de sensibilidade e todas as variáveis são não-negativas.

Neste método, a solução (caso exista) estará sempre na interceção de duas restrições. Para isso procura-se uma solução nas interceções. De seguida, segue-se até uma outra interceção, seguindo uma das restrições, que por normal é a que provoca a maior variação no valor da função objetivo. Uma solução inicial trivial é considerar que as variáveis de decisão são todas nulas. O algoritmo *Simplex* está ilustrado na Figura 6.

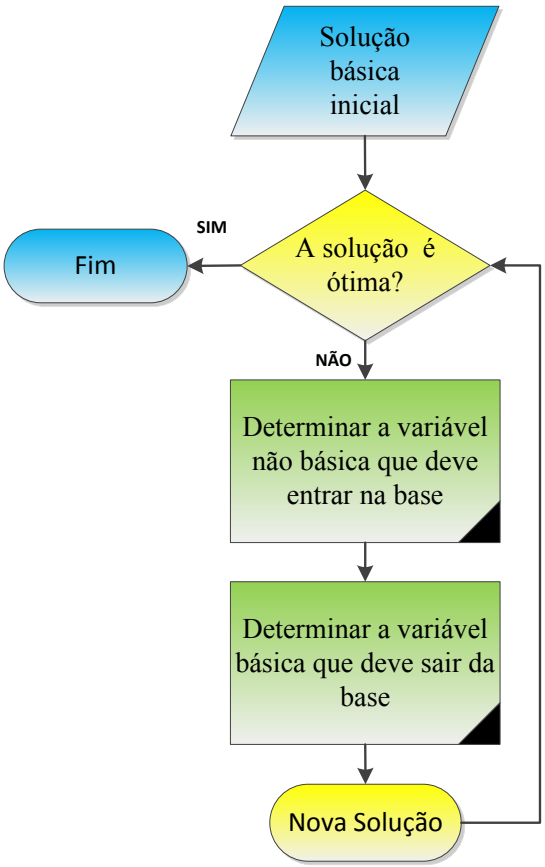

Figura 6 – Algoritmo Simplex.

Para mais detalhes sobre o método de *Simplex* poderá consultar [21].

3.1.2 Método da Relaxação Lagrangeana

A Relaxação Lagrangeana é uma técnica introduzida por Lagrange em 1797. A relaxação Lagrangeana tem como princípio um problema de programação inteira "difícil" inicialmente visto como fácil mas que se pode tornar complicado quando é adicionado um conjunto reduzido de restrições. Esta técnica baseia-se na decomposição das restrições, juntando-as à função objetivo através de um vetor de multiplicadores, chamados de multiplicadores de Lagrange, e eliminadas em seguida do conjunto de restrições deve produzir um problema Lagrangeano que é fácil de resolver e cujo valor da solução ótima é um limite inferior (para problemas de minimização) para o valor ótimo do problema original. O Problema Lagrangeano pode, portanto, ser usado no lugar de um problema de Relaxação Linear para produzir limites num algoritmo de procura do tipo *branch and bound*. Além disso, com base nesse limite inferior, é possível estimar a quão próxima está a solução viável disponível da solução ótima [22].

Desde 1970, esta tem sido a técnica de escolha para resolução de problemas de otimização. Esta técnica é principalmente usada para:

- *Bounding*: como se trata de otimização, é necessário haver um relaxamento, sendo o valor ótimo limitado pelo valor ideal do problema real.
- *Lagrangian heuristic*: gerar uma solução ótima a partir de "bom" com base numa solução para o problema relaxado.
- *Problem reduction*: reduzir o problema original baseado na solução para o problema relaxado.

Para dar início ao problema é necessário fazer a sua abordagem através da decomposição. Inicialmente, as restrições estrategicamente escolhidas são divididas em:

- "Boas", com a qual o problema pode ser resolvido muito facilmente.
- "Más" em que o tornam muito difícil de resolver.

Após a decisão das restrições a utilizar é necessário escolher taticamente os multiplicadores de Lagrange, dado que o relaxamento das restrições depende dos multiplicadores escolhidos.

3.2 Programação não Linear

Os problemas de Programação Não Linear (PNL) são definidos por um sistema de igualdades e desigualdades, denominadas restrições, sobre um conjunto de variáveis reais desconhecidos, juntamente com uma função objetivo a ser maximizada ou minimizada, onde alguns das restrições ou a função objetivo é não-linear. A diferença entre um problema de programação linear e um de programação não linear está no facto de que no problema não linear pelo menos uma das equações ser não linear, podendo mesmo ser a função objetivo.

A Programação Não-Linear Inteira Mista (MINLP) refere-se à programação matemática com variáveis contínuas e discretas e não-linearidades na função objetivo e restrições. Este tipo de abordagem é realizada quando é necessário otimizar simultaneamente a estrutura do sistema (discreto) e parâmetros (contínua).

Os MINLP têm sido utilizados em diversas aplicações, incluindo em sistemas industriais, engenharia e operações de investigação.

As necessidades em áreas tão diversas têm motivado a investigação e o desenvolvimento em tecnologia de forma a solucionar problemas MINLP, particularmente em algoritmos com problemas altamente combinatórios.

Os problemas MINLP são muito difíceis de resolver, porque são combinadas todas as dificuldades das suas subclasses: a natureza dos problemas inteiros (MIP) e a dificuldade em resolver problemas não-linear (PNL). As subclasses MIP e PNL estão entre as classes de, teoricamente, problemas difíceis (NP-completo), por isso não é de estranhar que a solução MINLP seja um processo desafiador e ousado [23].

3.3 Métodos heurísticos

3.3.1 Algoritmo Genético

O Método de algoritmo Genético (GA) é baseado numa população de representações abstratas de soluções candidatas para um problema de otimização que evolui em direção a melhores soluções. O GA é baseado nas operações de evolução, ou seja, de herança, de mutação, seleção e cruzamento [24].

Sendo assim, um algoritmo genético é uma técnica de programação que imita a evolução biológica como uma estratégia de resolução de problemas. Dado um problema específico para resolver, a entrada para o GA é um conjunto de soluções potenciais para este problema e uma função de aptidão que permite avaliar cada um dos candidatos quantitativamente.

No algoritmo, existem quatro operações básicas: cálculo de aptidão (*fitness evaluation)*, seleção (*selection*), cruzamento (*crossover*) e mutação (*mutation*). Ao fim destas operações cria-se uma nova população, chamada de geração (*generation*), que se espera representar uma melhor aproximação da solução ótima (*best solution)* do problema de otimização que a população anterior. A população inicial é gerada atribuindo-se aleatoriamente valores aos genes de cada cromossoma.

O algoritmo genético é ilustrado na Figura 7.

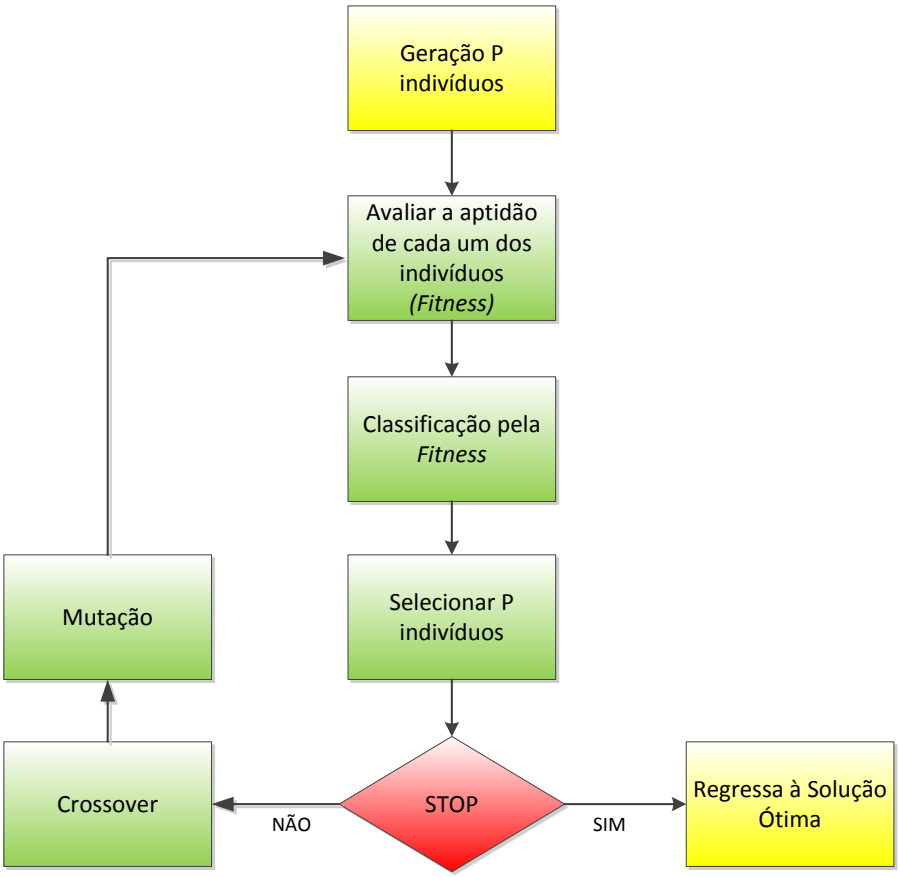

Figura 7- Algoritmo Genético.

Uma população de *P* indivíduos é gerada aleatoriamente. Cada um dos indivíduos da população representa uma possível solução para o problema, ou seja, um ponto no espaço de soluções.

Geralmente a **aptidão** do indivíduo é determinada através do cálculo da função objetivo, que depende das especificações. Cada indivíduo é uma entrada para uma função de análise de desempenho, cuja saída fornece medidas que permitem ao algoritmo genético o cálculo da aptidão do indivíduo. Ainda nesta fase os indivíduos são ordenados conforme a sua aptidão.

O problema para escrever a função de aptidão deve ser cuidadosamente considerado, de modo que maior aptidão é atingível e realmente se iguala a uma solução melhor para o problema dado. Se a função de aptidão é mal escolhida ou definida de forma imprecisa, o algoritmo genético pode ser incapaz de encontrar uma solução para o problema, ou pode-se resolver erradamente o problema.

A **seleção** determina os indivíduos mais aptos da geração atual e seleciona-os. Esses indivíduos são utilizados para gerar uma nova população por cruzamento. São

selecionados os *n* melhores indivíduos daquela geração e depois faz uma espécie de torneio. Neste método, denominado "amostragem universal estocástica", cada indivíduo tem uma probabilidade de ser selecionado proporcional à sua aptidão. A fase da seleção pode ser efetuada através de diferentes métodos de seleção de cromossomas, como por exemplo,

- Método da roleta: Coloca-se sobre este círculo uma "roleta" com *n* cursores, igualmente espaçados. Após uma volta da roleta a posição dos cursores indica os indivíduos selecionados. Isto significa que quanto melhores são os cromossomas, mais oportunidades têm de serem selecionados.
- Método da classificação: Primeiro é classificada a população e depois é atribuído a cada cromossoma um valor de adequação determinado pela sua classificação. O pior terá adequação igual a 1 e o melhor terá adequação igual a *N* (número de cromossomas na população). Este método de seleção pode resultar numa menor convergência pois os melhores cromossomas não se distinguem dos outros.

Os indivíduos selecionados na etapa anterior são cruzados (***crossover***). A forma como se realiza este cruzamento é ilustrada na Figura 8. Os cromossomas de cada par de indivíduos a serem cruzados são divididos num ponto de corte aleatoriamente. Um novo cromossoma é gerado permutando-se a metade inicial de um cromossoma com metade final do outro onde são recombinadas as características, criando dois novos indivíduos [25].

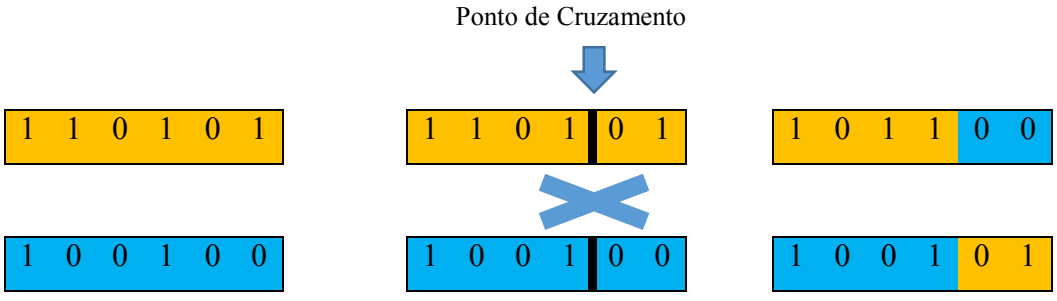

Figura 8 - *Crossover*.

A operação de **mutação** é efetuada alterando-se o valor de um gene de um indivíduo sorteado aleatoriamente com uma determinada probabilidade, denominada probabilidade de mutação, ou seja, vários indivíduos da nova população podem ter um dos seus genes alterado aleatoriamente [26].

Os GAs são aplicados a muitos problemas científicos, de engenharia, de negócios e de entretenimento, incluindo:

- *Otimização*: numa grande variedade de tarefas de otimização, incluindo otimização numérica, e os problemas de otimização combinatória, tais como o problema do caixeiro-viajante (TSP).
- *Programação automática*: desenvolvimento de *softwares* de programação para tarefas específicas, e para projetar outras estruturas computacionais.
- *Robot and Machine Learning*: aplicações de robôs a nível de conceção e de controlo, e também para criar redes neurais.
- *Modelos económicos*: modelação de processos de inovação, desenvolvimento de estratégias de licitação, devido ao aparecimento vários tipos de mercados económicos.
- *Modelos do sistema imunitário*: modelação de vários aspetos do sistema imunológico natural, incluindo mutação somática durante a vida de um indivíduo e da descoberta de vários 'genes' famílias durante o tempo evolutivo.
- *Modelos ecológicos*: modelação em fenómenos ecológicos, tais como raças biológicas, parasitas-hospedeiro, coevoluções, simbiose e fluxo de recursos em ecologias.
- *Modelos de genética de populações*: estudo das questões de genética de populações, tais como "em que condição é que um gene recombinado pode ser evolutivamente viável?"
- *Interações entre evolução e aprendizagem*: estudo da forma como a aprendizagem individual e a evolução das espécies se afetam mutuamente.
- *Modelos de sistemas sociais*: estudo dos aspetos evolutivos dos sistemas sociais, tais como a evolução da cooperação, o comportamento e a evolução da comunicação.

O algoritmo genético prova ser bastante eficiente na solução de problemas de otimização, garantido convergir à solução ótima, utilizando uma população de indivíduos que se submetem a seleção na presença de variação de indução, tais como operadores de mutação e de recombinação.

Um problema bem conhecido que pode ocorrer com uma GA é conhecido como convergência prematura. Se um indivíduo que está mais em forma do que a maioria dos seus concorrentes emerge no início do curso da execução, ele pode reproduzir de forma tão abundante que derruba a diversidade da população muito cedo, levando o algoritmo a convergir para um ótimo local [24-25].

Finalmente, vários investigadores, tais como [22] e [24], advertem contra o uso de algoritmos genéticos em problemas que podem ser resolvidos analiticamente. Não é que os algoritmos genéticos não possam encontrar boas soluções para esses problemas, é apenas que os métodos tradicionais de análise utilizam muito menos tempo e esforço computacional do que o GA que garante a solução exata, embora se considere que ainda não há nada que encontre uma solução matematicamente perfeita para qualquer problema de adaptação biológica.

Fazendo uma analogia, ver Tabela 1, entre o processo biológico e o processo de otimização, pode verificar-se que, e.g. um individuo representa uma possível solução e uma geração representa um ciclo no processo de otimização.

Tabela 1 – Analogia da Natureza

Analogia com a Natureza		
Evolução Natural	⇔	**Algoritmos Genéticos**
Indivíduo	—	Solução
Genótipo (cromossomas)	—	Representação da Solução
Reprodução Sexual	—	Operador de Recombinação (p.ex. cruzamento)
Mutação	—	Operador Mutação
População	—	Conjunto de Soluções
Gerações	—	Ciclos

Os GAs procuram iterativamente a melhor solução a partir de conjuntos de pontos de partida, tal como já foi definido através do Algoritmo GA (verificar Figura 7).

3.3.2 *Particle Swarm Optimization* (PSO)

Swarm Intelligence (SI) é um método inovador para resolução de problemas de otimização que originalmente tiveram a sua inspiração em sistemas biológicos por

enxame, bandos e pastorícia. O *Particle Swarm Optimization* (PSO), ou otimização por enxame de partículas, é um conceito que foi introduzido para a otimização não-linear utilizando as estratégias inspiradas pelo comportamento das colónias, como por exemplo, enxames, cardumes, bandos e até mesmo comportamentos humanos, iterativamente [30].

Este método passou por muitas mudanças desde a sua introdução em 1995 [31]. Os investigadores desde essa data têm desenvolvido novas aplicações e publicados estudos sobre os vários parâmetros e aspetos do algoritmo.

O PSO é uma meta-heurística, dado que não se baseia, ou quase não se baseia em suposições sobre o problema que está a ser otimizado e pode pesquisar espaços alargados de soluções candidatas. No entanto, meta-heurísticas como PSO não garantem que a solução ideal é sempre encontrada, pois não usa a informação da derivada das funções do problema a ser otimizado, o que significa que o PSO não requer que o problema a otimizar seja diferenciável como é exigido por métodos clássicos de otimização.

Para aplicação de PSO com sucesso, uma das questões-chave é encontrar a forma de delinear a solução do problema na partícula PSO, o que afeta diretamente a sua viabilidade e desempenho.

A variante básica do algoritmo PSO funciona por ter uma população (chamado de enxame) de soluções candidatas (chamado partículas). Estas partículas são movimentadas no espaço. Um algoritmo básico PSO é ilustrado na figura que se segue.

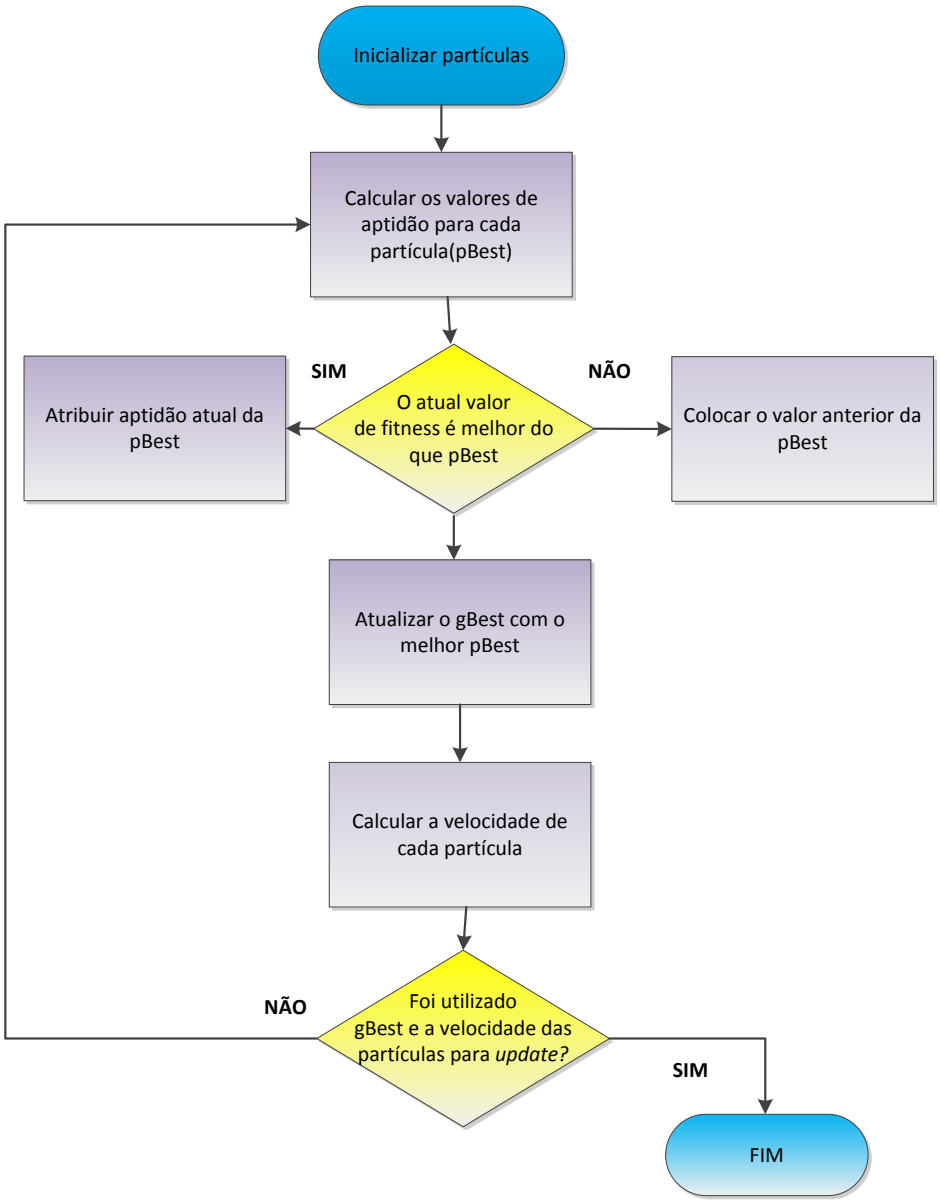

Figura 9- Algoritmo PSO.

Mais detalhes relativos ao método *Particle swarm optimization* podem ser consultados em [32].

3.3.3 Ant Colony Optimization

Desde sempre que as formigas fascinaram o ser humano, pela sua forma de percorrer caminhos na procura de comida, a forma como faziam as suas reservas, como se refugiam quando são interrompidas por obstáculos e da forma como contornam os problemas que lhes aparecem ao longo do caminho.

Pelo seu comportamento em colónia foram, observadas por investigadores que pretenderam descobrir mais sobre a sua história e a forma como vivem. Foi assim que mais de perto foram observadas. As primeiras tentativas detalhadas e amplamente reconhecida para estudar o comportamento das formigas para a pesquisa científica foram feitas no final de 1980. As formigas comunicam-se através de rastos de feromonas (substância química de reconhecimento, utilizada para sinalização).

A experiência de Goss [33] consistia num conjunto de formigas reais, em que as formigas depositavam feromonas enquanto caminham de uma fonte de alimento até ao ninho, e vice-versa (ver Figura 10). Durante as experiências foram usados dois cenários, um que tinha dois caminhos com a mesma distância e um segundo que tinha um caminho mais longo que outro. Constatou-se que, sensivelmente 50% das formigas, no primeiro cenário, escolheram um dos caminhos, enquanto que no segundo caso esse número aumentou significativamente para o caminho mais curto. Com esta simples experiencia verificou-se que as formigas são ótimas fontes de inspiração para problemas que visam encontrar o caminho mais curto entre dois pontos.

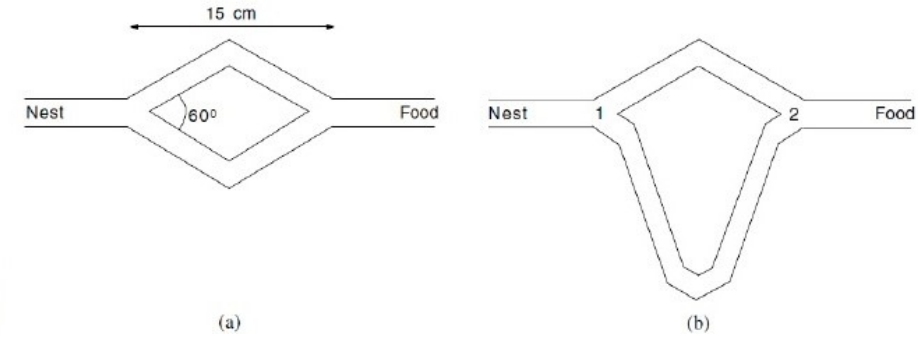

Figura 10- Experiência de Goss.

Depois de terem sido realizados testes com formigas reais foram criadas formigas artificiais de modo a dar continuidade aos estudos. As formigas artificiais constroem as soluções de forma incremental e probabilística, através de duas formas:

- Trilhos artificiais de feromonas (matriz τ), que representa uma "memória" sobre o processo de procura;
- Informação heurística (matriz η), que representa a informação a priori sobre o problema que será resolvido.

Surge então a meta-heurística ACO (*Ant Colony Optimization*) [34] que é um método de otimização, analisando o comportamento de uma colónia de formigas, pelo caminho que percorrem, com o intuito de o otimizar. O método faz parte do novo conjunto de algoritmos de otimização chamado *Swarm Intelligence*.

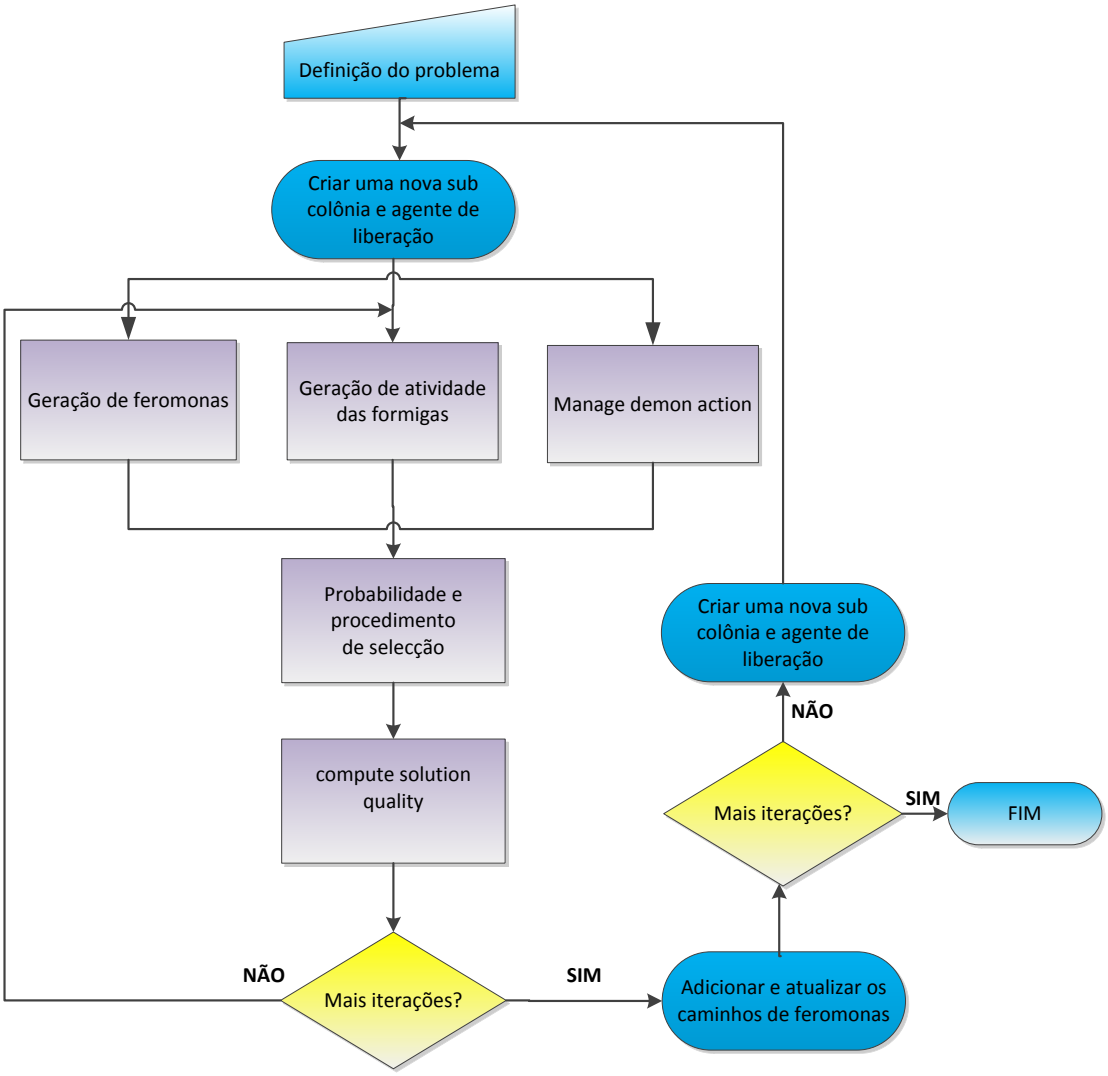

Figura 11- Algoritmo ACO.

O algoritmo ACO tem muitas vantagens na sua utilização (Figura 11):

- Aplicável em vários de problemas do setor industrial.
- Os problemas podem ter uma dimensão elevada.
- O algoritmo é muito flexível e é passível de ser desenvolvido através de uma série de linguagens de programação e *softwares*, incluindo plataformas de código aberto. A maioria das organizações possui plataformas em que os algoritmos ACO podem ser executados.

3.3.4 Tabu Search

Para resolver de forma genérica problemas de otimização são utilizadas técnicas meta-heurísticas. Estas técnicas são usadas em situações que podem ser modeladas como problemas de maximizar (ou minimizar) uma função cujas variáveis tem certas restrições.

As meta-heurísticas são estratégias utilizadas para resolver problemas complexos difíceis por oferecerem melhores soluções e geralmente com tempo de processamento menor do que por outros tipos de técnicas.

De forma geral, utilizam combinação de escolhas aleatórias e conhecimento histórico (dos resultados anteriores adquiridos pelo método) para se guiarem e realizar suas procuras pelo espaço de pesquisa em vizinhanças dentro do espaço de pesquisa, o que evita paragens prematuras em ótimos locais.

Tabu Search é uma meta-heurística que faz uma pesquisa na vizinhança de forma 'inteligente' explorando o espaço de soluções além da otimização local, de uma forma eficaz. O processo de procura é acelerado usando métodos aleatórios de seleção. Consiste basicamente numa técnica de melhoria da solução, que considera estruturas que permitam explorar eficientemente o histórico de todo o processo de procura [31-32].

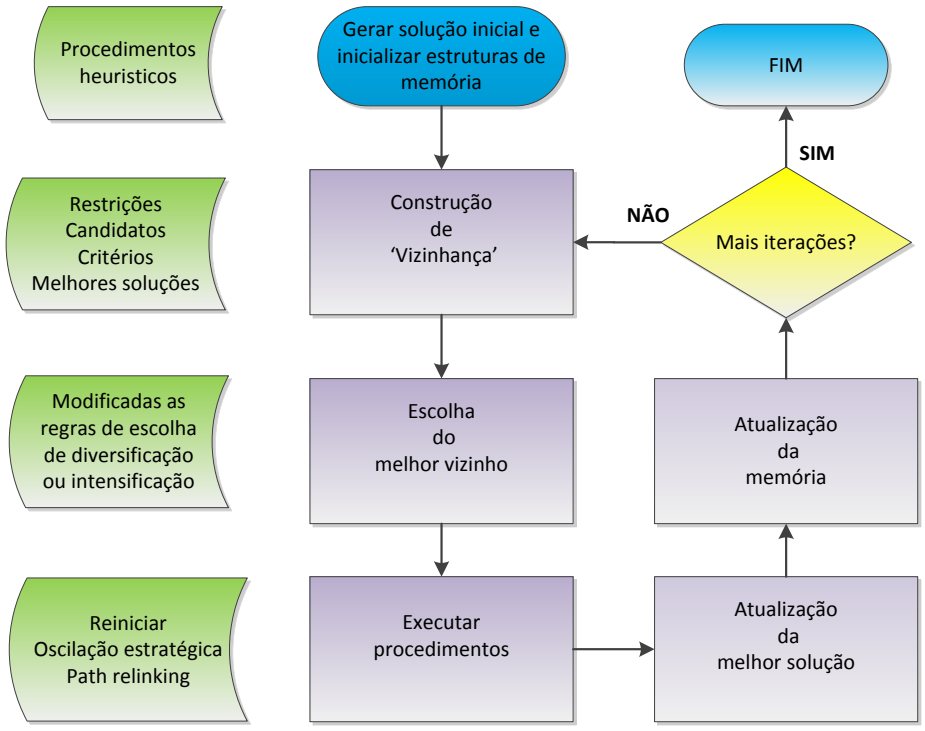

Figura 12- Algoritmo *Tabu Search*.

As restrições do *Tabu Search* não são invioláveis em todas as circunstâncias e são utilizados vários tipos de memórias, tanto de curto prazo e a longo prazo, de modo a melhorar a qualidade da exploração.

3.3.5 Simulated Annealing

A origem da técnica de otimização conhecida por *Simulated Annealing* (arrefecimento simulado), vem de 1953 [37], quando foi usada para simular num computador o processo de arrefecimento de cristais e foi descrita por Kirkpatrick e a sua equipa em 1983 [38].

O arrefecimento de alguns materiais consiste em submetê-los numa fase inicial a altas temperaturas e reduzi-las gradualmente até atingirem, com aumentos e reduções do estado de energia, o equilíbrio térmico, tornando-os assim, consistentes e rígidos.

A técnica matemática de *Simulated Annealing* é um algoritmo que procura de uma forma iterativa o próximo candidato a ponto de mínimo na vizinhança do candidato atual, agindo de acordo com a diferença entre os valores da função objetivo. Esta técnica evita mínimos locais usando uma procura aleatória que, por vezes, aceita pontos que podem ter valores maiores para a função objetivo, fazendo que, em algumas iterações, o algoritmo tenda a maximizar a função objetivo em vez de minimizá-la [39].

3.4 Métodos Determinísticos

3.4.1 Método dos pontos interiores

O método de pontos interiores, popularizado por *Karmarkar* [40] para a resolução de problemas não lineares, necessita de um ponto viável interior (relativo) disponível. A partir daí, novos pontos interiores são gerados numa vizinhança de uma trajetória central (até se atingir uma certa tolerância para uma solução ótima). De forma a atingir-se a solução ótima, deve-se realizar um procedimento de purificação de uma solução, isto é, eliminação dos pontos de não interesse. Em termos de complexidade, os algoritmos do tipo pontos interiores é considerado polinomial [40].

3.4.2 Método *Nelder-Mead*

O método *Nelder-Mead*, foi proposto por *John Nelder* e *Roger Mead* em 1965, é uma técnica de otimização não-linear bastante utilizada uma vez que não necessita da informação das derivadas das funções que constituem o problema de otimização. O método utiliza o conceito de um *simplex* com $N + 1$ vértices, onde N representa a dimensão do espaço de procura.

3.5 Métodos populacionais vs. métodos ponto-a-ponto

Outra forma de classificação dos métodos de otimização é segundo a sua técnica de procura da solução. De um lado podemos encontrar métodos baseados em pesquisa por população e de outros métodos de pesquisa baseados num único ponto.

Os métodos baseados em pesquisa por população são do tipo teste e erro e não do tipo força bruta, i.e. várias soluções podem ser encontradas antes de se chegar a uma solução admissível. Por norma, até se chegar a uma solução, são encontradas soluções intermédias nas quais, a sua qualidade, i.e. aproximação à solução ótima, aumenta gradualmente. A grande diferença entre os dois grupos, reside no que o próprio nome indica, que se prende por uma pesquisa por parte de um conjunto de soluções paralelas nos métodos populacionais, enquanto que nos métodos ponto-a-ponto, só uma solução em cada iteração é encontrada.

Alguns exemplos de métodos populacionais são os métodos de algoritmos genéticos, *particle swarm optimization* e *ant colony optimizations*. Os métodos *tabu search*, *simulated annealing* e pontos interiores são métodos ponto-a-ponto.

Capítulo 4

Caso de estudo

A aplicabilidade de técnicas emergentes de escalonamento, como sejam os algoritmos genéticos, foi testada usando a célula de fabrico AIP-PRIMECA localizada na *Université de Valenciennes et du Hainaut-Cambrésis* (UVHC), França.

4.1 Descrição do Caso de Estudo

A célula AIP-PRIMECA é composta por 7 estações de trabalho interligadas através de um sistema monocarril (ver Figura 13).

Figura 13- Visão geral da célula AIP-PRIMECA.

A primeira estação de trabalho está encarregue de carregar e descarregar as placas de base, a serem transportadas, num *shuttle* (Figura 14).

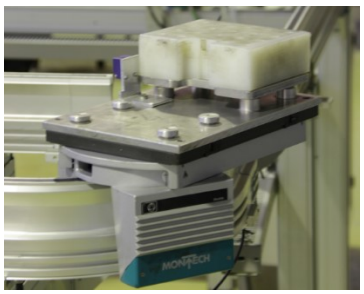

Figura 14 – *Shuttle* de transporte.

As estações de trabalho dois, três e quatro utilizam robôs Kuka que permitem a colocação e montagem de componentes automaticamente, sendo que cada robô é capaz de montar três tipos de componentes neste sistema de fabrico flexível. Cada componente pode ser colocado numa de duas estações de trabalho diferentes. Esta redundância introduz flexibilidade no sistema, o que é particularmente útil em casos de mau funcionamento do robô.

A quinta estação de trabalho é uma zona de controlo visual automatizado, com recolha das imagens dos produtos, verificando o estado de produtos acabados.

A sexta estação de trabalho é uma unidade de reparação manual sendo utilizada no caso de deteção de defeito na zona de trabalho 6.

A sétima estação de trabalho é uma zona de trabalho que não é utilizada atualmente. De futuro irá servir para poder aumentar a flexibilidade da célula através da oferta de mais serviços.

Tanto o recurso 6 como o recurso 7 não são usados nesta implementação, i.e., assume-se, para o caso do recurso 6 que os produtos são inspecionados com sucesso e o recurso 7 não é usado por opção.

Como já referido anteriormente, o sistema de transporte é um sistema de monocarril, onde as junções ou bifurcações são obtidas através de "portas" rotativas. Desta maneira, este sistema de transporte pode ser considerado como um grafo orientado, fortemente ligado, composto dos seguintes nós (Figura 15).

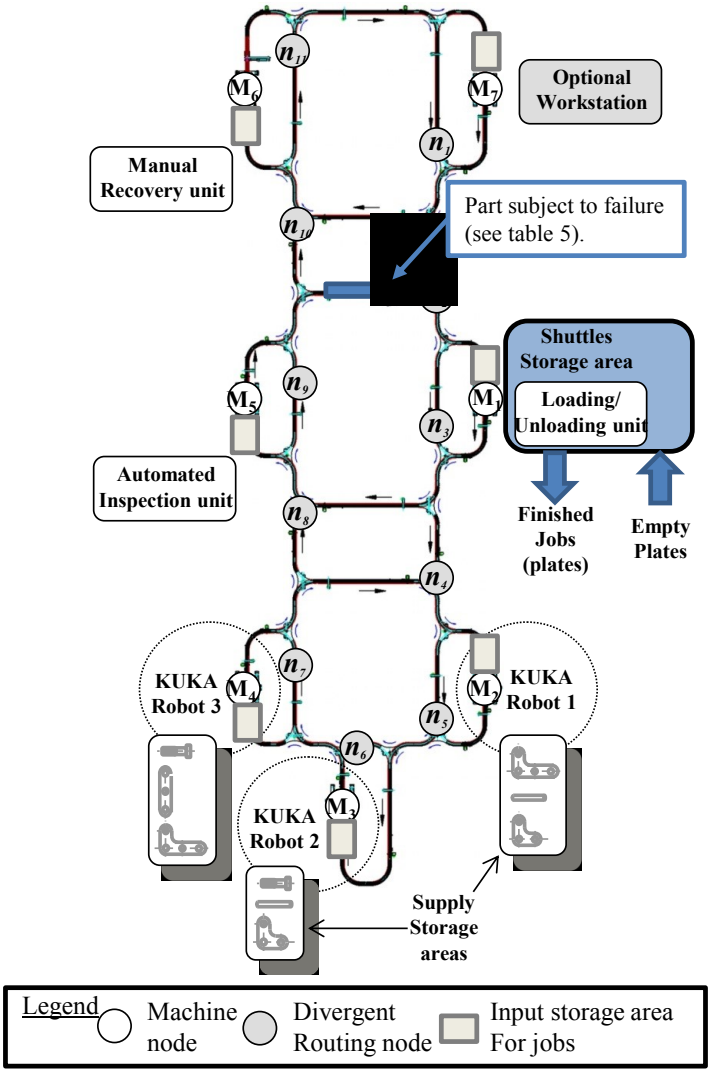

Figura 15 – Recursos, nós e layout do sistema [41]

Na figura anterior, M_1, M_2, M_3, M_4, M_5, M_6, M_7 representam as máquinas, e n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}, n_{11} representam os nós de roteamento divergentes em que as decisões de encaminhamento devem ser feitas (por exemplo, a partir de n_{11}, é possível alcançar a adjacente nodos n_1 ou M_7).

O sistema de fabrico é capaz de produzir três produtos distintos, sendo cada um composto por subprodutos. Os 3 produtos a serem produzidos são: BELT, AIP e LATE, sendo que o produto BELT, por exemplo, é constituído pelos subprodutos B, E, L e T. Então, por exemplo, para uma encomenda de 10 BELT, terão de ser produzidos 10 subprodutos de cada uma das letras.

Cada subproduto segue um plano de produção (*process plan*), que é composto por tarefas elementares: "Plate_loading", "Axis_comp", "I_comp", "L_comp", "r_comp", "Screw_comp", "Plate_unloading" e "Inspection".

Assim, a Tabela 2, resume a ordem e as tarefas necessárias a executar para a manufatura de cada subcomponente.

Tabela 2- Plano de realização dos componentes

sequence	*B*	*E*	*L*	*T*	*A*	*I*	*P*
#1	Loading	Loading	Loading	Loading	Loading	Loading	Loading
#2	Axis	Axis	Axis	Axis	Axis	Axis	Axis
#3	Axis	Axis	Axis	Axis	Axis	Axis	Axis
#4	Axis	Axis	Axis	r_comp	Axis	I_comp	r_comp
#5	r_comp	r_comp	I_comp	L_comp	r_comp	Screw_comp	L_comp
#6	r_comp	r_comp	I_comp	Inspection	L_comp	Inspection	Inspection
#7	I_comp	L_comp	Screw_comp	Unloading	I_comp	Unloading	Unloading
#8	Screw_comp	Inspection	Screw_comp		Screw_comp		
#9	Inspection	Unloading	Inspection		Inspection		
#10	Unloading		Unloading		Unloading		

A Tabela 3 resume os diversos serviços oferecidos pelo sistema, assim como os tempos necessários em cada recurso para os executar.

Tabela 3 – Tempos dos serviços/máquina

	M1	*M2*	*M3*	*M4*	*M5*	*M6*
Loading	*10*					
Unloading	*10*					
Axis		*20*	*20*			
r_comp		*20*	*20*			
I_comp				*20*		
L_comp		*20*		*20*		
Screw_comp			*20*	*20*		
Inspection					*5*	
Recovery						*60*

Este problema pode ser descrito através de um modelo matemático que formalize as relações, restrições e função objetivo. Neste trabalho, esta modelação foi baseada no trabalho descrito em [41], e encontra-se descrito na secção que se segue.

4.2 Elaboração do Modelo Matemático para o Caso de Estudo

Para a resolução deste problema é necessário resolver um problema de minimização com restrições, definido por:

$$C_{max} = min(t_{ln})\ \forall i\ \in I_j, \forall j \in P \qquad [.1]$$

onde *l* é a última operação da tarefa *n*.

Para a definição das restrições é necessário definir os seguintes parâmetros/variáveis. Assim, consideremos:

- *J* conjunto de tarefas, J = 1, 2,…., n

- P conjunto de tarefas a serem realizadas: P = 1, 2,….,n
- R Conjunto de máquinas, R = 1, 2,….., r
- I_j Conjunto de operação da tarefa $_j$, I_j = 1, 2,…., $|Ij|$, j ∈ J
- O_{ij} Operação i da tarefa j
- M_{ij} conjunto de máquinas possíveis para a operação O_{ij}
- p_{ij} Tempo de processamento da operação i ($i \in Ij$)
- MJ Número máximos de trabalhos ao mesmo tempo
- tt_{m1m2} tempo de transporte da máquina m_1 para m_2
- c_{ir} capacidade de entrada da fila na máquina r
- d_j Prazo da tarefa j, j = 1;…., n
- αj, βj são os atrasos e multas dos atrasos associado ao trabalho j, j = 1;……, n

As **variáveis do problema** são definidas da seguinte forma:

- t_{ij} *tempo* de conclusão da operação t_{ij} O_{ij} ($_i \in I_j$), $t_{ij} \in N$.
- μ_{ijr} um conjunto variável binária: 1 se a operação O_{ij} é realizada na máquina r, 0, caso contrário.
- b_{ijkl} um conjunto variável binária :1 se a operação O_{ij} é realizada antes da operação O_{kl}, 0 caso contrário.
- tr_{ijr1r2} um conjunto variável binária: 1 se j trabalho é transportado para máquina r_2 após a realização de operação O_{ij}, 0 caso contrário.
- w_{ijr} tempo de espera de funcionamento O_{ij} na fila da máquina r.
- wv_{ijklr} um conjunto variável binária: 1 se a operação O_{ij} está à espera de O_{kl} operação na fila da máquina r, 0, caso contrário
- z_{lj} definido para 1 se trabalho l e trabalho j estão na prontos ao mesmo tempo, 0 caso contrário.

Relativamente às restrições podemos definir:

<u>Restrições disjuntivas</u>

Uma máquina pode processar uma operação em cada instante e a operação é realizada por uma única máquina.

$$t_{ij} + p_{kl} + BM * b_{ijkl} \leq t_{kl} + BM \quad \forall i,k \in I, \forall j,l \in P, \forall r \in R_{ij,} \qquad [2]$$

onde BM assume um valor elevado

$$b_{ijkl} + b_{klij} \leq 1 \; \forall i \in I_j, k \in I_l, \forall j,l \in P \qquad [3]$$

$$\sum \mu_{ijr} = 1 \; \forall i \in I_j, \forall j \in P \qquad [4]$$

Onde *i, j, k* e *l* são operações e *r* as máquinas.

Restrições de precedência

Garantir a sequência de tarefas de um produto. A conclusão da operação seguinte considera a conclusão da operação precedente, o tempo de espera e o tempo de transporte, se as duas operações não são realizadas na mesma máquina.

$$t_{(i+1)j} \geq t_{ij} + p_{(i+1)j} + w_{(i+1)jr2} + \sum_{r1r2} tt_{r1r2} \, tr_{ijr1r2} \; \forall i \in I_j, \forall j \in P, \forall r_{1,}r_2 \in R_{ij} \qquad [5]$$

$$\sum_{\substack{r1,r2 \in R \\ r1 \neq r2}} tr_{ijr1r2} \leq 1 \forall i \in I_j, \forall j \in P$$

Relação alocação e transporte

Se as operações sucessivas de um produto são realizados em máquinas diferentes, isso implica a existência de uma operação de transporte entre as duas máquinas. Atrasos de transporte são ajustados para zero e o sistema de transporte tem uma capacidade ilimitada.

$$\mu_{ijr1} + \mu_{(i+1)jr2} - 1 \leq tr_{ijr1r2} \; \forall i \in I_j, \forall j \in P, \forall r_{1,}r_2 \in R_{ij}, r_1 \neq r_2 \qquad [6]$$

$$\mu_{ijr1} + \mu_{(i+1)jr2} \geq (1 + e) \, tr_{ijr1r2} \; \forall i \in I_j, \forall j \in P, \forall r_{1,}r_2 \in R_{ij}, r_1 \neq r_2, \qquad [7]$$

onde *e* é um número muito baixo

Capacidade da entrada na fila da máquina e regra FIFO (*first in first out*)

Cada máquina tem uma capacidade máxima de entrada na fila, sendo que quando cheia não pode receber mais *shuttles*. A ordem de chegada determina a ordem de operações, i.e., o primeiro a chegar é o primeiro a ser executado.

$$b_{ijkl} + w_{ijklr} \leq 1 \; \forall i,k \in I, \forall j,l \in P, \forall r \in R_{ij} \cap R_{kl} \qquad [8]$$

$$b_{ijkl} - w_{ijklr} \geq 0 \; \forall i \in Ij, \; \forall k \in Il, \forall j,l \in P, \forall r \in R_{ij} \cap R_{kl} \qquad [9]$$

$$wv_{ijklr} + wv_{klijr} \leq 1 \; \forall i \in Ij, \; \forall k \in Il, \forall j, l \in P, \forall r \in R_{ij} \cap R_{kl} \quad [10]$$

$$t_{ij} - p_{ij} + BM * b_{ijkl} + BM * wv_{klijr} \leq t_{kl} - p_{kl} - w_{klr} + 2*BM \quad [11]$$
$$\forall i \in Ij, \; \forall k \in Il, \forall j, l \in P, j \neq l, \forall r \in R_{ij} \cap R_{kl}$$

$$w_{klr} \leq \sum_{\substack{i \in I \\ j \in P, j \neq 1}} pijwvklijr \; \forall k \in Il, \forall l \in P, \forall r \in R_{kl} \quad [12]$$

$$\mu_{ijr} + \mu_{klr} \geq 2 \, (wv_{ijklr} + wv_{klijr}) \; \forall i \in Ij, \; \forall k \in Ik, \forall j, l \in P, \forall r \in R_{ij} \cap R_{kl} \quad [13]$$

$$t_{ij} + BM* b_{ijkl} \leq t_{kl} + BM \; \forall i \in Ij, \; \forall k \in Ik, \forall j, l \in P \quad [14]$$

$$t_{ij} + p_{ij} \; \mu_{ijr} - w_{ijr} + BM* b_{ijkl} \leq t_{kl} - p_{kl} \; \mu_{klr} - w_{klr} + BM \quad [15]$$
$$\forall i \in Ij, \; \forall k \in Ik, \forall j, l \in P, \forall r \in R$$

$$\sum wv_{ijklr} \leq ci_r - 1 \; \forall i \in Ij, \; \forall j \in P, \forall r \in R_{ij} \cap R_{kl} \quad [16]$$

A limitação do número de produtos no sistema

O número de trabalhos a serem executados em simultâneo no mesmo *shop floor* pode ser limitada por MJ.

$$\sum z_{lj} \leq MJ - 1 \; \forall j \in P \quad [17]$$

$$z_{lj} \geq b_{0luj} + b_{0l0j} \text{ -1 } \forall j, l \in P \quad [18]$$

$$z_{lj} \leq 1\text{- } b_{0l0j} + b_{ujul} \; \forall j, l \in P \quad [19]$$

$$z_{lj} \geq b_{0l0j} + b_{ujul} \text{ -1} \forall j, l \in P \quad [20]$$

Restrições para cada tipo de variável

$$t_{ij} > p_{ij} \; \forall i \in Ij, \; \forall j \in P \quad [21]$$

$$b_{ijkl} \in \{0,1\}\ \forall i \in Ij,\ \forall j \in P, \forall k \in Il, \forall l, \in P \quad [22]$$

$$tr_{ijr1r2} \in \{0,1\}\ \forall i \in Ij,\ \forall j \in P, \forall r_1, r_2 \in R_{ij} \quad [23]$$

$$\mu_{ijr} \in \{0,1\}\ \forall i \in Ij,\ \forall j \in J, \forall r \in R_{ij} \quad [24]$$

No capítulo seguinte serão apresentados alguns detalhes de implementação desta formulação matemática, assim como uma análise aos resultados obtidos.

Capítulo 5

Resultados numéricos

Neste capítulo é descrita a implementação do modelo matemático descrito no capítulo anterior e analisados os resultados obtidos para alguns cenários. Neste estudo, foi escolhido o método de algoritmos genéticos para implementar o escalonamento de produtos da célula AIP-PRIMECA.

Para o efeito, foi utilizado o *software* Matlab™, que se trata de uma linguagem de alto nível e um ambiente interativo de programação e visualização. O Matlab™ permite o desenvolvimento de algoritmos matemáticos e a sua posterior análise, assim como a criação de modelos próprios e o desenvolvimento de aplicações. A linguagem é matematicamente acessível e o seu conjunto de ferramentas permite explorar várias abordagens e chegar a uma solução mais rápida. Como método de otimização, foi usado o método de algoritmos genéticos através da função GA do Matlab™ (existente na *toolbox GA Optimset*).

5.1 Função predefinida do Matlab™ - GA

O *software* Matlab™ traz já entre as suas bibliotecas o algoritmo baseado em GA. Esta função tem como parâmetros de entrada, entre outros, a função objetivo, o número de variáveis do problema e as suas restrições. Como saídas, a função devolve, entre outros, o melhor ponto obtido durante as iterações, o valor da função no ponto anterior e uma *flag* que indica a razão de saída do algoritmo.

Além das entradas e saídas, é possível controlar alguns parâmetros relacionados com a função. Por exemplo, é possível definir a função usada como técnica de *crossover*, definir o número de gerações a usar e o valor da população a usar durante o algoritmo.

5.2 Implementação do Modelo Matemático

A implementação do modelo matemático compreendeu a codificação das equações descritas no capítulo anterior usando o *software* Matlab™. De forma a não tornar esta descrição longa, esta secção exemplifica a codificação de algumas dessas equações.

Como ficheiro de controlo da otimização, i.e. onde se introduzem os dados relativos aos produtos a serem desenvolvidos no sistema de fabrico (modelado pela formulação matemática descrita anteriormente) é apresentado na seguinte implementação.

Implementação 1: Dados do problema

```
op=[3; 3; 3; 4; 5];

[nlcineq,nlceq]=nonlcon1I(op)

x0=[op(:,1)']
A=[]
b=[]
Aeq=[]
beq=[]
lb=[1;1;1;1;1]
ub=[5;5;5;5;5]
nx0=length(x0);

for i=1:10
   [x,fval,exitflag,output]=ga(@funI,5,A,b,Aeq,beq,lb,ub,@nonlcon1I)
end;
```

De notar que na implementação anterior é descrita a sequência de operações, os limites inferiores e superiores a usar na função GA, assim como as restrições que descrevem o problema (@nonIcon1I).

A implementação da função objetivo descreve-se na seguinte implementação.

Implementação 2: Função objetivo

```
function y=funI(x)
op(:,1)=[round(x(1:2))]
op(:,2)=[round(x(3:4))]
 [b,p,tt,t,miu,tr,w,z,r,BM,MJ,C,e,wv]=valuesparametersI(op);
[n,m]=size(op)
 y=max(t(n,:));
end
```

Na implementação pode-se verificar a utilização da função *valuesparametersI*, que tem como entrada o vetor *op*. Nessa função são descritas algumas restrições que dizem respeito à sequência de operações necessárias para a realização de um produto, tal como as precedências.

Por exemplo, a equação 7, que indica se duas operações consecutivas necessitam de transporte entre recursos, é codificada da seguinte forma:

Implementação 3: Equação 7

```
for i=1:n-1
    for j=1:m
        a=op(i,j);
        a1=op(i+1,j);
        nlcineq=[nlcineq; miu(i,j,a)+miu(i+1,j,a1)-1-tr(i,j,a,a1)];
    end
end
```

De notar que para esta restrição, usa-se uma outra, *miu*, que indica se uma operação é realizada no recurso (ver notação do problema).

Outro exemplo de implementação, que descreve a relação de precedências de operações num recurso, com o tempo de espera nesse mesmo recurso, é apresentada na seguinte implementação.

Implementação 4: Equação 8

```
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                a=op(i,j);
                nlcineq=[nlcineq; b(i,j,k,l)+wv(i,j,k,l,a)-1];
            end
        end
    end
end
```

Finalmente, o exemplo que se segue ilustra a relação dos tempos de conclusão de operações precedentes, os tempos de execução da operação em causa, com a necessidade, ou não, de transporte entre recursos.

Implementação 5: Equação 15

```
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                 nlcineq=[nlcineq; t(i,j)+BM*b(i,j,k,l)-t(k,l)-BM];
            end
        end
    end
end
```

De realçar que a implementação anterior utiliza os parâmetros por defeito para a *toolbox* do GA.

Convém ainda salientar que, de forma a simplificar a implementação do problema, algumas restrições não foram implementadas, nomeadamente as restrições descritas nas equações 5 e 12.

5.3 Definição de Cenário

O cenário em estudo no presente trabalho passou pela realização de um produto do tipo "I", existente no catálogo de produtos possíveis de serem produzidos na célula. Na definição do cenário considerou-se também que os tempos de transporte entre recursos era negligenciável. Além disso, reduziu-se o número de operações necessárias para a produção do produto, eliminando as operações de "loading" e "unloading".

5.4 Resultados obtidos com a função GA

Numa primeira fase, foi testada a formulação matemática implementada recorrendo à implementação proprietária do Matlab™ do método GA. A Tabela 4 resume os resultados obtidos.

Tabela 4 – Resultados de diversas simulações

Opções	Time	fval	
Plot Fcns	1989	-	No feasible point
Pop Size	1868	-	No feasible point
tolcon	1702	-	No feasible point
Tolfun	1083	-	No feasible point
hibrydFcn	801	-	No feasible point
Crossover	827	-	No feasible point

Após várias simulações, alterando diversos parâmetros da simulação, constatou-se que nenhuma combinação conseguiu chegar a uma solução admissível, obtendo-se *'Optimization terminated: no feasible point found.'*. Desta forma, pode-se concluir, que com esta formulação matemática o algoritmo GA proprietário do Matlab™ é incapaz de chegar a uma solução admissível. Tal deve-se ao facto do problema em causa possuir um número elevado de restrições e a implementação do GA não conseguir identificar, em nenhuma iteração, um ponto admissível.

De forma a ultrapassar esta situação, optou-se por fazer uma implementação dinâmica do método de algoritmos genéticos, que designámos por GA-JS.

5.5 Resultados obtidos com a implementação GA-JS

Como não se conseguiu obter uma solução admissível com a abordagem anterior, foi usado uma implementação dinâmica do algoritmo GA [42], que não usa qualquer ferramenta da *toolbox* do Matlab™. Esta implementação difere da anterior no sentido que apenas efetua pesquisa aleatória em subconjuntos possíveis da variável op(i). Assim, é possível eliminar um elevado conjunto de restrições definidas na seção 4.2. Para mais detalhes, consultar [43].

Com a implementação GA-JS já é possível identificar pontos admissíveis.

Uma vez que o método GA é estocástico, foram efetuadas 30 execuções e a tabela seguinte apresenta o valor mínimo obtido nas 30 execuções ($C_{max}^{min}(s)$), a da solução obtida ($C_{max}^{avg}(s)$) e o tempo médio de execução de cada execução ($time^{avg}$); todos os resultados são dados em segundos (s). A solução fornecida por este método é dada na Tabela 5.

De facto, utilizando uma implementação modificada do algoritmo GA, é possível chegar a soluções admissíveis, tendo sido elaborados estudos mais detalhados que consideram cenários com uma maior complexidade de produtos, descritos em [43] e ilustrados na Tabela 5.

Tabela 5 – Resultados obtidos com GA-JS [43]

	$C_{max}^{min}(s)$	$C_{max}^{avg}(s)$	$time^{avg}(s)$
AIP	170	182	14,5
BELT	240	261	16,3
LATE	230	248	16,3

Após a análise da tabela, é possível constatar que, por exemplo, para a realização um lote AIP, o tempo médio de execução do algoritmo é de 14,5s.

Para o mesmo caso, o C_{max} médio foi de 182s, sendo a melhor solução encontrada de 170s [43].

5.6 Conclusões

O modelo matemático apresentado em [41], e que constituiu a base da formulação elaborada no capítulo 4, possui muitas restrições que dificilmente são satisfeitas simultaneamente. Isto torna-se evidente quando se usa o algoritmo proprietário do

Matlab™, uma vez que este não consegue chegar a uma solução admissível. Mesmo quando, exaustivamente, se variam diversos parâmetros do algoritmo de otimização, este nunca conseguiu encontrar uma solução admissível.

De forma a contornar este problema, usou-se uma versão modificada do GA, com a qual foi possível encontrar soluções admissíveis [43], sendo que neste caso, as soluções encontradas para os casos descritos, demoram entre 14,5s e 16,3s.

Capítulo 6

Conclusões e Trabalho Futuro

As técnicas de escalonamento, nomeadamente as baseadas em meta-heurísticas, assumem um papel cada vez mais importante no escalonamento de sistemas complexos, como são os que se encontram nos sistemas de fabrico.

O presente trabalho classificou alguns tipos de problemas existentes em sistemas de fabrico e classificou algumas técnicas que podem ser usadas como meios de otimização para poder realizar o escalonamento nos anteriores.

Foi implementada uma formulação matemática, baseado numa modelação existente, para uma célula de produção real existente na *Université de Valenciennes et du Hainaut-Cambrésis*, usando o *software* Matlab™. Depois desta implementação, um problema de escalonamento, neste caso para o produto "I", foi executado usando a técnica baseada em algoritmos genéticos existente no Matlab™. Verificou-se que com esta implementação não se conseguiu identificar nenhuma solução admissível, devendo-se ao facto do número de restrições do problema e, provavelmente, à forma genérica como a implementação do GA é realizada no Matlab™.

Numa segunda fase do trabalho, optou-se por usar uma abordagem de uma implementação diferente do GA, que já conseguiu obter soluções admissíveis para o problema. Esta nova implementação, designada por GA-JS, forneceu diversos pontos admissíveis tendo identificado a solução ótima entre 14,5 e 16,3 segundos.

Como trabalho futuro recomenda-se a reformulação da modelação matemática, corrigindo algumas das restrições, e a aplicação de técnicas diferentes, baseadas em meta-heurísticas, como por exemplo o método *particle swarm*, *simulated annealing* ou *tabu search.*

Referências

[1] P. Kotler and G. Armstrong, *Principles of marketing*, 15th ed. Upper Saddle, N.J: Pearson, 2006.

[2] M. P. Groover, *Automation, production systems, and computer-integrated manufacturing*, 3rd ed. Upper Saddle River, N.J: Prentice Hall, 2008.

[3] *http://www.manutencaoesuprimentos.com.br/conteudo/3988-sistema-de-manufatura-flexivel/*. .

[4] P. Leitão, "*An Agile and Adaptive Holonic Architecture for Manufacturing Control*," PhD, *University* of Porto, 2004.

[5] D. Gerwin, "*Do's and Don'ts of Computerized Manufacturing*," *Harvard Business Review*, pp. 107–116, 1982.

[6] R. Jaikumar, *Flexible Manufacturing Systems: A Managerial Perspective*, Harvard Business School. Boston, 1984.

[7] S. E. Garrett, "*Strategy First: A Case in FMS Justification," presented at the ORSA/TIMS Conference on Flexible Manufacturing Systems*, 1986, pp. 17–30.

[8] *Cambridge University Press, "Scheduling," 2013.*

[9] *WebFinance, Inc, "Scheduling," 2013.*

[10] *E. Moradi and M. Zandieh, "Minimizing the makespan and the system unavailability in parallel machine scheduling problem: a similarity-based genetic algorithm*," *The International Journal of Advanced Manufacturing Technology*, vol. 51, no. 5–8, pp. 829–840, Jun. 2010.

[11] H. Huang, "*Component-based design for job-shop scheduling systems*," *The International Journal of Advanced Manufacturing Technology*, vol. 45, no. 9–10, pp. 958–967, Apr. 2009.

[12] D. B. Shmoys, C. Stein, and J. Wein, "*Improved Approximation Algorithms for Shop Scheduling Problems*," *SIAM Journal on Computing*, vol. 23, no. 3, pp. 617–632, Jun. 1994.

[13] F. Pezzella, G. Morganti, and G. Ciaschetti, "*A genetic algorithm for the Flexible Job-shop Scheduling Problem*," *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, Oct. 2008.

[14] H. Fisher and G. L. Thompson, "*Probabilistic learning combinations of local job-shop scheduling rules*," *Industrial Scheduling*, pp. 225–251, 1963.

[15] R. Qing-dao-er-ji and Y. Wang, "*A new hybrid genetic algorithm for job shop scheduling problem*," *Computers & Operations Research*, vol. 39, no. 10, pp. 2291–2299, Oct. 2012.

[16] C.-Y. Lee, L. Lei, and M. Pinedo, "*Current trends in deterministic scheduling*," *Annals of Operations Research*, vol. 70, no. 0, pp. 1–41, 1997.

[17] L. Portilho, "Aplicação do Problema de Job-Shop Scheduling em Aciarias," Universidade Federal de Juiz de Fora, 2007.

[18] S. P. Boyd, *Convex optimization*. Cambridge, UK ; New York: Cambridge University Press, 2004.

[19] A. S. Jain and S. Meeran, "*Deterministic job-shop scheduling: Past, present and future*," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, Mar. 1999.

[20] G. B. Dantzig and M. N.-D. Thapa, *Theory and extensions*. New York: Springer, 1997.

[21] K. G. Murty, *Linear programming*. New York: Wiley, 1983.

[22] M. Held and R. Karp, "*The Traveling Salesman Problem and Minimum Spanning Trees*," *Operations Research*, pp. 1138–1162, 1970.
[23] M. Bussieck and A. Pruessner, "*Mixed-Integer Nonlinear Programming.*" *GAMS Development Corporation*, 2003.
[24] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, 1st MIT Press ed. Cambridge, Mass: MIT Press, 1992.
[25] M. Caraciolo, *Artificial Intelligence in Motion*, 2009.
[26] M. Obikto, *Genetic Algorithms*.
[27] S. Forrest, "*Genetic algorithms: principles of natural selection applied to computation*," *Science*, vol. 261, no. 5123, pp. 872–878, Aug. 1993.
[28] M. Mitchell, *An introduction to genetic algorithms*. Cambridge, Mass.: MIT Press, 1998.
[29] R. L. Haupt, *Practical genetic algorithms*, 2nd ed. Hoboken, N.J: John Wiley, 2004.
[30] R. Poli, J. Kennedy, and T. Blackwell, "*Particle swarm optimization*," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, Aug. 2007.
[31] J. Kennedy and R. Eberhart, "*Particle swarm optimization*," vol. 4, pp. 1942–1948.
[32] G. Venter and J. Sobieszczanski-Sobieski, "*Particle Swarm Optimization*," *American Institute of Aeronautics and Astronautics*, vol. 41, no. 8, pp. 1583–1589, 2003.
[33] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels, "*Self-organized shortcuts in the Argentine ant*," *Naturwissenschaften*, vol. 76, no. 12, pp. 579–581, Dec. 1989.
[34] M. Dorigo, "*Optimization, Learning and Natural Algorithms*," Politecnico di Milano, Italy, 1992.
[35] É. D. Taillard, "*Parallel Taboo Search Techniques for the Job Shop Scheduling Problem*," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 108–117, May 1994.
[36] H. R. Lourenço and M. Zwijnenburg, "*Combining the Large-Step Optimization with Tabu-Search: Application to The Job-Shop Scheduling Problem*," in *Meta-Heuristics*, I. H. Osman and J. P. Kelly, Eds. Boston, MA: Springer US, 1996, pp. 219–236.
[37] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "*Equation of State Calculations by Fast Computing Machines*," *The Journal of Chemical Physics*, vol. 21, no. 6, p. 1087, 1953.
[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "*Optimization by Simulated Annealing*," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
[39] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "*Job Shop Scheduling* by Simulated Annealing," *Operations Research*, vol. 40, no. 1, pp. 113–125, Feb. 1992.
[40] N. Karmarkar, "*A new polynomial-time algorithm for linear programming*," 1984, pp. 302–311.
[41] D. Trentesaux, C. Pach, A. Bekrar, Y. Sallez, T. Berger, T. Bonte, P. Leitão, and J. Barbosa, "*Benchmarking flexible job-shop scheduling and control systems*," *Control Engineering Practice*, vol. 21, no. 9, pp. 1204–1225, Sep. 2013.
[42] A. Curralo, A. I. Pereira, J. Barbosa, and P. Leitão, "*Flexible Job Shop Scheduling Problem in Manufacturing*," presented at the XVI Congresso da Associação Portuguesa de Investigação Operacional, 2013, pp. 70–71.
[43] A. Curralo, A. I. Pereira, J. Barbosa, and P. Leitão, "*Sensibility study in a flexible job shop scheduling problem*," 2013, pp. 634–637.

Anexos

Dados do problema a otimizar

```
op=[3; 4; 3; 4; 5];

[nlcineq,nlceq]=nonlcon1I(op)

x0=[op(:,1)']
A=[]
b=[]
Aeq=[]
beq=[]
lb=[1;1;1;1;1]
ub=[5;5;5;5;5]
nx0=length(x0);


for i=1:1
 [xx,y,exit, nonl]=ga(@funI,5,A,b,Aeq,beq,lb,ub,@nonlcon1I)
  inf(i,1)=exit;
     inf(i,2)=y;
    inf(i,3:nx0+2)=x0;
end;
```

Definição da função

```
function y=funI(x)

op(:,1)=[round(x(1:2))]
op(:,2)=[round(x(3:4))]



[b,p,tt,t,miu,tr,w,z,r,BM,MJ,C,e,wv]=valuesparametersI(op);
[n,m]=size(op)

y=max(t(n,:));
end
```

Restrições

```
function [nlcineq,nlceq]=nonlcon1I(x)

op(1:5,1)=[round(x(1:5))];
[b,p,tt,t,miu,tr,w,z,r,BM,MJ,C,e,wv]=valuesparametersI(op);
[n,m]=size(op); % n operation ; m job ; r machine number


%linequality constraints (<=0)
```

```
nlcineq=[];
%4.2
disp('Restrição 4.2')

for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                a=op(i,j);
                nlcineq=[nlcineq;
t(i,j)+p(k,l)*miu(k,l,a)+BM*b(i,j,k,l)-t(k,l)-BM];
            end
        end
    end
end
%4.3%
disp('Restrição 4.3')
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                nlcineq=[nlcineq; b(i,j,k,l)+b(k,l,i,j)-1];
            end
        end
    end
end
%4.5%

disp('Restrição 4.5')
for i=1:n-1
    for j=1:m
        a=op(i,j);
        a1=op(i+1,j);
        nlcineq=[nlcineq; -
t(i+1,j)+t(i,j)+p(i+1,j)+w(i+1,j,a1)+(tt(a,a1)*tr(i,j,a,a1))];
    end
end

%4.6
disp('restrição 4.6')
for i=1:n-1
    for j=1:m
        a=op(i,j);
        a1=op(i+1,j);
        nlcineq=[nlcineq; tr(i,j,a,a1)-1];
    end
end
%4.7%
disp('Restrição 4.7')
for i=1:n-1
    for j=1:m
        a=op(i,j);
        a1=op(i+1,j);
        nlcineq=[nlcineq; miu(i,j,a)+miu(i+1,j,a1)-1-tr(i,j,a,a1)];
    end
end
%4.8
disp ('Restrição 4.8')
for i=1:n-1
    for j=1:m
        for k=1:n
```

```
            for l=1:m
                a=op(i,j);
                a1=op(i+1,j);
                nlcineq=[nlcineq; -miu(i,j,a)-
miu(i+1,j,a1)+(1+e)*tr(i,j,a,a1)];
            end
        end
    end
end
%4.9%
disp('Restrição 4.9')
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                a=op(i,j);
                nlcineq=[nlcineq; b(i,j,k,l)+wv(i,j,k,l,a)-1];
            end
        end
    end
end
%4.10%
disp('Restrição 4.10')
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                a=op(i,j);
                nlcineq=[nlcineq; wv(i,j,k,l,a)-b(i,j,k,l)];

            end
         end
    end
end
%4.11%
disp('Restrição 4.11')
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                a=op(i,j);
                nlcineq=[nlcineq; wv(i,j,k,l,a)+wv(k,l,i,j,a)-1];
            end
         end
    end
end
%4.12%
disp('Restrição 4.12')
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                a=op(i,j);
                nlcineq=[nlcineq; t(i,j)-
p(i,j)+BM*b(i,j,k,l)+BM*wv(k,l,i,j,a)-t(k,l)+p(k,l)+w(k,l,a)-2*BM];

            end
        end
    end
end
%4.13
```

```
disp ('restrição4.13')

for i=1:n
    for j=1:m
        a=op(i,j);
         nlcineq=[nlcineq; w(k,l,a)-(p(i,j)*wv(k,l,i,j,a))];
    end
end

%4.14%
disp('Restrição 4.14')
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                a=op(i,j);
                nlcineq=[nlcineq; -miu(i,j,a)-
miu(k,l,a)+2*(wv(i,j,k,l,a)+wv(k,l,i,j,a))];

            end
        end
    end
end

%4.15%
disp('Restrição 4.15')
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                 nlcineq=[nlcineq; t(i,j)+BM*b(i,j,k,l)-t(k,l)-BM];
            end
        end
    end
end
%4.16%
disp('Restrição 4.16')
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                a=op(i,j);
                nlcineq=[nlcineq; t(i,j)-p(i,j)*miu(i,j,a)-
w(i,j,a)+BM*b(i,j,k,l)-t(k,l)+p(k,l)*miu(k,l,a)+w(k,l,a)-BM];
            end
        end
    end
end
%4.17%
disp('Restrição 4.17')
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                a=op(i,j);
                nlcineq=[nlcineq;wv(i,j,k,l,a)-C(i,a)+1];

            end

        end
```

```
    end
end
%4.18%
disp('Restrição 4.18')
for j=1:m
    for l=1:m
           nlcineq=[nlcineq; z(l,j)-MJ+1];
    end
end

%4.22%
disp('Restrição 4.22')
for i=1:n
    for j=1:m
           nlcineq=[nlcineq; p(i,j)-t(i,j)];
    end
end


%nonlinear equality constraints (=0)
nlceq=[];
%4.4%
for i=1:n
    for j=1:m
        a=op(i,j);
            nlceq=[nlceq; miu(i,j,a)-1];
        end
    end
end
```

Definição de parâmetros

```
function [t,miu,tr,w,z,wv] = values1(op,p,r)

[n,m]=size(op)

% t definition
t=zeros(n,m)
t(1:n,1:m)
p(1:n,1:m)
t(1,:)=p(1,:)
for i= 1:n-1
    for j=1:m-1
        t(i+1,i)=t(i,j)+p(i+1,i)%+w
t(i,j)=p(i,j)
t(i+1,j+1)=t(i,j)+p(i,j)%+w

    end;
end;

% miu definition
miu(1:n,1:m,1:n,1:m)=0;
for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
            a=op(i,j);
            miu(i,j,a)=1
```

```
            end;
        end;
    end;
end;

% tr definition
tr(1:n,1:m,1:r,1:r)=0;
for i=1:n-1
    for j=1:m
        a=op(i,j);
        a1=op(i+1,j);
        tr(i,j,a,a1)=0
    end;
end;

% w definition
w(1:n,1:m,1:r)=0;
for j=1:m
    for i=2:n
    w(i,j,op(i,j))=t(i-1,j) % ver com os resultados
    end;
end;
% z definition
z(1:n,1:m)=0;

% wv definition
wv(1:n,1:m,1:n,1:m,1:r)=0;

for i=1:n
    for j=1:m
        for k=1:n
            for l=1:m
                for rr=1:r
                    if (op(i,j)==op(k,l))&&(op(i,j)==rr)
                        wv(i,j,k,l,rr)=1;
                    else wv(i,j,k,l,rr)=0;
                    end;
                end;
            end;
        end;
    end;
end;


end


function [b,p,tt,t,miu,tr,w,z,r,BM,MJ,C,e,wv]=valuesparametersI(op)

% b definition
b=zeros(5,1,5,1);
b(1,1,2,1)=1; b(2,1,3,1)=1; b(4,1,5,1)=1;b(3,1,4,1)=1;

% p definition
p=[20;20;20;20;5];

% tt definition
tt=[0 5 5 5 0;0 5 5 5 0;0 5 5 5 0;0 5 5 5 0;0 5 5 5 0];
```

```
% r definition
n=5;m=1;
r=5;w=2;BM=10;MJ=2;C(1:n,1:r)=0;e=0.01;


op=[3; 3; 3; 4; 5];
job1
%Loading
%%Axis_comp
%op(1,1).maq=[2,3];
%op(2,1).maq=[2,3];
%I_comp
%op(4,1).maq=[3,4];
%Inspection
%op(5,1).maq=[5];

[t,miu,tr,w,z,wv] = values1(op,p,r);

end
```