ANÁLISE DO DESEMPENHO DE TÉCNICAS DE OTIMIZAÇÃO NO PROBLEMA DE ESCALONAMENTO

André Borges Guimarães Serra e Santos

Departamento de Engenharia Mecânica

Instituto Superior de Engenharia do Porto

2015

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha da Disciplina de Dissertação / Projecto / Estágio, do 2º ano, do Mestrado em Engenharia Mecânica - Gestão Industrial

Candidato: André Borges Guimarães Serra e Santos, Nº 1070953, 1070953@isep.ipp.pt

Orientação científica: Ana Maria Dias Madureira Pereira, amd@isep.ipp.pt

Coorientação científica: Maria Leonilde Rocha Varela, leonilde@dps.uminho.pt

Departamento de Engenharia Mecânica

Instituto Superior de Engenharia do Porto

4 de Novembro de 2015

Agradecimentos

Não posso deixar de apresentar os meus profundos agradecimentos aos que contribuíram na realização deste trabalho, onde destaco a minha orientadora, Professora Doutora Ana Madureira, pelo extraordinário apoio científico, confiança que demostrou e disponibilidade e paciência que sempre manifestou, bem como, a minha coorientadora, Professora Doutora Leonilde Varela, que esteve sempre disponível para esclarecer as duvidas e incentivar de foram entusiástica e otimista.

Saliento também o apoio incondicional da minha família.

Resumo

A optimização nas aplicações modernas assume um carácter fortemente interdisciplinar, relacionando-se com a necessidade de integração de diferentes técnicas e paradigmas na resolução de problemas reais complexos. O problema do escalonamento é recorrente no planeamento da produção. Sempre que uma ordem de fabrico é lançada, é necessário determinar que recursos serão utilizados e em que sequência as atividades serão executadas, para otimizar uma dada medida de desempenho. Embora ainda existam empresas a abordar o problema do escalonamento através de simples heurísticas, a proposta de sistemas de escalonamento tem-se evidenciado na literatura.

Pretende-se nesta dissertação, a realização da análise de desempenho de Técnicas de Optimização, nomeadamente as meta-heurísticas, na resolução de problemas de optimização complexos – escalonamento de tarefas, particularmente no problema de minimização dos atrasos ponderados, $1||\Sigma wjTj$. Assim sendo, foi desenvolvido um protótipo que serviu de suporte ao estudo computacional, com vista à avaliação do desempenho do *Simulated Annealing* (SA) e o *Discrete Artificial Bee Colony* (DABC).

A resolução eficiente de um problema requer, em geral, a aplicação de diferentes métodos, e a afinação dos respetivos parâmetros. A afinação dos parâmetros pode permitir uma maior flexibilidade e robustez mas requer uma inicialização cuidadosa. Os parâmetros podem ter uma grande influência na eficiência e eficácia da pesquisa. A sua definição deve resultar de um cuidadoso esforço experimental no sentido da respectiva especificação. Foi usado, no âmbito deste trabalho de mestrado, para suportar a fase de parametrização das meta-heurísticas em análise, o planeamento de experiências de Taguchi.

Da análise dos resultados, foi possível concluir que existem vantagem estatisticamente significativa no desempenho do DABC, mas quando analisada a eficiência é possível concluir que há vantagem do SA, que necessita de menos tempo computacional.

Palavras-Chave

Escalonamento, Meta-Heurísticas, *Simulated Annealing* (SA), *Discrete Artificial Bee Colony* (DABC), Taguchi.

Abstract

Optimization in its modern applications has strong interdisciplinary character, which is related to the necessity of integrating diverse techniques to approach complex problems. Scheduling is an important part of the production planning. Each time a production order is released, it is necessary to determine the resources that will be employed and in what sequence the tasks will be executed, to enhance a certain performance measure. While some enterprises still approach the scheduling problem with simple heuristic rules, advanced, automatized scheduling problems are becoming more popular.

In this dissertation, the performance of the Simulated Annealing (SA) and the Discrete Artificial Bee Colony (DABC), in the resolution of the minimization of the weighted tardiness scheduling problem ($\Sigma w_j T_j$), were studied. In order to compare the performance of the two meta-heuristics, both were implemented in a software prototype.

Even with the development of several meta-heuristics, including the Simulated Annealing (SA) and the Discrete Artificial Bee Colony (DABC), there is still uncertainty on what makes some meta-heuristics more efficient than others, without a comparative analyses. Moreover, meta-heuristics parameters impact on how the techniques will perform, on how effective and efficient is the exploration of solution space, while also requiring a careful revision. In this dissertation, the meta-heuristics were tuned with Taguchi experiments, which allow a comprehensive and efficient revision of the parameters.

Through the analysis of the results, it was possible to conclude about a relevant statistical performance advantage of the DABC in the resolution of the minimization of the weighted tardiness problem ($\Sigma w_j T_j$), however, SA appeared more efficient, since it required less computational time than DABC in most instances of the problem.

Keywords

Scheduling, Meta-Heuristics, Simulated Annealing (SA), Discrete Artificial Bee Colony (DABC), Taguchi.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

ABC – Artificial Bee Colony

ACO – Ant Colony Optimization

AIS – Artificial Immune Systems

AS – Ant System

ATC – Apparent Tardiness Cost

BCO – Bee Colony Optimization

CA – Cultural Algorithms

DABC – Discrete Artificial Bee Colony

DOE – Design of Experiments

DSS – Decision Support System

GA – Genetic Algorithms

GRASP – Greedy Adaptive Search Procedure

HS – Harmony Search

ILS – Iterative Local Search

IP – Integer Programming

LNS – Largest Number of Successors

LPT – Longest Processing Time

MT – Movimento de Tarefa

MBO – Marriage in Honey Bees Optimization

MCT – Minimum Completion Time

MET – Minimum Execution Time

MOMCT – Modified Ordered Minimum Completion Time

LS – Local Search

OX – Order Crossover

ParamILS – Parameter Iterated Local Search

PMX – Partially Mapped Crossover

PSO – Particle Swarm Optimization

QBE – Queen-Bee Evolution

REVAC – Relevance Estimation and Value Calibration

SA – Simulated Annealing

SPO – Sequential Parameter Optimization

SPT – Shortest Processing Time

TT – Troca de Tarefas

TTA – Troca de Tarefas Adjacentes

TS – Tabu Search

VBA – Virtual Bee Algorithm

VNS – Variable Neighborhood Search

WSPT – Weighted Shortest Processing Time

WSPT-MCI – Weighted Shortest Processing Time First with Minimum Cost Insertion

1. INTRODUÇÃO

O planeamento e controlo da produção, que deverá determinar como irá decorrer o processo produtivo, determina como as ordens de fabrico serão executadas, isto é, quem deverá executar as tarefas e quando deverão ser executadas, para maximizar uma medida de desempenho. É então o departamento de planeamento e controlo da produção que realiza o escalonamento, distribuindo as tarefas pelos recursos e determinando quando, ou em que sequência, as tarefas devem ser executadas. O correto escalonamento das operações minimiza os tempos produtivos e os atrasos, permitindo rentabilizar os recursos produtivos produzindo mais rapidamente e com custos mais reduzidos.

A função de escalonamento deverá ter em consideração as particularidades do sistema produtivo, as características dos recursos e a forma como estão dispostos na oficina. Por exemplo, não é necessário distribuir as tarefas num sistema produtivo onde devem ser executadas todas na mesma máquina e não é necessário sequenciar as tarefa em problemas de minimização do tempo de processamento, em máquinas paralelas.

O desenvolvimento da teoria da complexidade computacional demostrou que os problemas não podem ser abordados todos por técnicas semelhantes. Os problemas de otimização fáceis podem ser resolvidos otimamente, enquanto os problemas difíceis rapidamente se tornam impraticáveis. O escalonamento de operações, particularmente quando envolvem decisões de sequenciamento, é extremamente complexo, porque o número de soluções de um problema de sequenciamento cresce exponencialmente em função da dimensão do problema. Isto torna impossível analisar todas as soluções do problema, sendo necessário recorrer a técnicas que não necessitem de enumerar a totalidade de soluções.

Existem inúmeras formas de abordar o escalonamento, desde técnicas construtivas, que são habitualmente muito fáceis de implementar, até técnicas de pesquisa, que exploram uma parte das soluções. Entre as técnicas mais utilizadas no problema de escalonamento estão as meta-heurísticas, que são técnicas aproximativas evoluídas, que fazem uma exploração orientada das soluções. Ao contrário da pesquisa de vizinhança, as meta-heurísticas são dotadas de mecanismos que permitem deslocar a pesquisa para áreas mais promissoras do espaço de soluções, sendo atualmente as técnicas mais usadas para abordar problemas de otimização complexos.

O conceito de meta-heurística assenta na pesquisa diversificada, que posteriormente se intensifica em torno das soluções promissoras, normalmente inspiradas em fenómenos observados na natureza, tornando-as particularmente interessantes em problemas onde existem muitas soluções e pouco tempo para determinar uma solução.

Embora exista muito interesse nas meta-heurísticas, o seu funcionamento não é totalmente compreendido. É impossível determinar à partida, que meta-heurística vai obter o melhor desempenho num determinado problema, sendo habitualmente escolhidas empiricamente, com base em preferência ou em resultados anteriores. O impacto dos parâmetros no desempenho das meta-heurísticas também é evidente e habitualmente a parametrização representa um esforço muito considerável. Esta falta de entendimento acerca do funcionamento das meta-heurísticas torna a análise comparativa, absolutamente essencial na altura de selecionar uma.

1.1. OBJECTIVOS

É consensual que as meta-heurísticas apresentam o melhor desempenho em problemas mais complexos. Mesmo muito utilizadas, ainda não é bem compreendido o que torna certas meta-heurísticas superiores. Assim, neste trabalho de mestrado, pretende-se:

- Analisar as características do problema de escalonamento, apresentando as medidas de desempenho e ambientes de fabrico. Fazer um levantamento dos principais problemas e das técnicas de otimização mais apropriadas.

- Analisar as características das meta-heurísticas, apresentar as principais meta-heurísticas e analisar os seus parâmetros, acompanhadas por um pequeno exemplo ilustrativo que permita perceber o seu funcionamento.

- Implementar duas meta-heurísticas e fazer uma análise do impacto dos parâmetros no problema de minimização dos atrasos ponderados. Determinar os parâmetros que resultam no melhor desempenho das meta-heurísticas em análise.

- Fazer o estudo computacional e analisar o desempenho das meta-heurísticas. Examinar o funcionamento das meta-heurísticas nos problemas testados, para poder identificar as principais diferenças na evolução da pesquisa.

- Realizar a análise estatística dos resultados das meta-heurísticas. Identificar a meta-heurística com melhores resultados no estudo computacional e inferir acerca do desempenho das meta-heurísticas em problemas idênticos.

1.2. ORGANIZAÇÃO DO RELATÓRIO

O relatório encontra-se divido em 5 capítulos. O 1º capítulo, é a introdução do relatório. O 2º capítulo, aborda o problema do escalonamento, desde da representação usada, aos ambientes de fabrico e às técnicas utilizadas para abordar o escalonamento. O 3º capítulo, foca-se nas meta-heurísticas, apresentando em detalhe 6 meta-heurísticas muito utilizadas e uma técnica de parametrização. O 4º capítulo, apresenta o estudo computacional e a análise estatística do desempenho das meta-heurísticas implementadas. Finalmente, no 5º capítulo, é apresentada a conclusão do trabalho e as ideias para trabalhos futuros.

2. ESCALONAMENTO DAS OPERAÇÕES

Neste capítulo serão apresentados os conceitos fundamentais do Escalonamento de Operações (EO). Serão descritos todos os elementos que constituem o problema de escalonamento, os tipos de implantação industriais, as restrições e objectivos deste importante processo de decisão. Será ainda apresentada a representação adoptada neste trabalho, bem como as técnicas utilizadas para abordar o EO. Finalmente, serão abordadas as principais dificuldades inerentes ao problema do escalonamento, que tornam este problema tão interessante, tanto do ponto de vista académico como industrial.

2.1. ENQUADRAMENTO HISTÓRICO

Durante o século XIX ocorreu uma significativa transformação na forma como os produtos eram fabricados. Nesta nova sociedade industrial, a produção artesanal foi substituída pelas primeiras unidades produtivas, altamente especializadas. Estas fábricas rudimentares, que fabricavam um pequeno número de produtos em enormes quantidades, recorriam a operários pouco instruídos coordenados por um chefe de oficina. Era da responsabilidade do chefe de oficina o planeamento, a coordenação e o controlo do processo de fabrico, incluindo a contratação de colaboradores ou a aquisição de matérias-primas.

O Escalonamento de Operações limitava-se à elaboração de um plano de trabalhos, definido empiricamente pelo chefe de oficina, que determinava quando cada ordem de fabrico devia começar a ser executada ou concluída. O planeamento das tarefas era bastante simples e não determinava o tempo total de execução de uma ordem de fabrico, nem fixava os tempos de execução necessários para as operações.

Em finais do século XIX, o incremento da complexidade das unidades produtivas, que rapidamente aumentavam a variedade de produtos a executar, tornou necessário alterar a função de planeamento da produção. No início do século XX, Taylor propõe uma separação da função de planeamento e da execução, através de um departamento de planeamento da produção que deveria controlar os inventários e monitorizar as operações. Era, ainda, o departamento de planeamento que deveria controlar o fluxo de fabrico, lançando encomendas. Gantt, em 1916 propõem o desenvolvimento de uma lista das tarefas que devem ser realizadas num dado dia, que o departamento de planeamento da produção deveria calendarizar para serem implementadas pelo chefe de oficina, que continuava a intervir ativamente no plano, através de alterações na sequência e alocação do trabalho. O Escalonamento de Operações proposto por Gantt utilizava além da quantidade de produtos, o tempo para controlar o desenrolar das operações.

O tratamento do Escalonamento de Operações como problema de otimização, com um impacto profundo no desempenho das unidades produtivas, acontece em meados do século XX. Durante os anos 60, formulações em programação dinâmica e programação inteira são elaboradas para otimizar o Escalonamento de Operação. Durante os anos 70, os desenvolvimentos de Karp acerca da Complexidade Computacional [1], conduziram à hierarquização dos problemas de escalonamento pela sua dificuldade. Durante os anos 80, o Escalonamento das Operações começou ser abordado através de técnicas de otimização, de aproximação e computação evolutivas, e nos anos 90 surgem os primeiros sistemas de escalonamento informáticos. Já no século XXI muitos dos desenvolvimentos na teoria do Escalonamento estão relacionadas com sistemas de escalonamento dinâmico, sistemas de escalonamento distribuídos e sistemas de escalonamento colaborativos, que permitem várias entidades colaborar no elaboração do plano de trabalhos.

O Escalonamento de Operações é atualmente uma área interdisciplinar onde se interligam trabalhos, no âmbito da Gestão Industrial, Ciências de Computação e Matemática [2,3].

2.2. PROBLEMA DE ESCALONAMENTO

O Escalonamento de Operações é preponderante para a sobrevivência de uma empresa, e pode ser definido como um processo de decisão que procura rentabilizar os recursos. O mercado é atualmente caracterizado pela sua extrema volatilidade e competitividade, onde clientes procuram produtos personalizados em prazos muito curtos, o que colocou uma enorme pressão sobre as empresas para melhor rentabilizar os seus recursos, de maneira a satisfazer as necessidades do cliente de uma forma eficiente.

O problema do escalonamento pode ser definido como uma função de decisão que distribui os recursos disponíveis pelas operações, durante um período de tempo [2]. Num contexto industrial, o Escalonamento de Operações pode ser inserido na cadeia de planeamento, onde acrescenta o detalhe final às ordens lançadas, como pode ser analisado na figura 1. É realizado na iminência das ordens serem lançadas, uma vez conhecidos os volumes e prazos de produção e determina como as operações serão executadas.

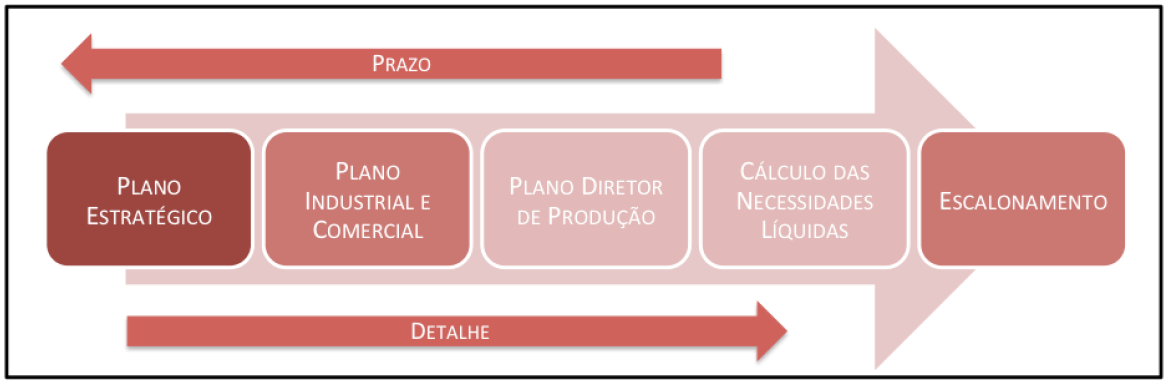

Figura 1 - Enquadramento do Escalonamento das Operações

Em Pinedo [4], o processo de escalonamento é descrito como uma função de planeamento que existe de uma forma mais ou menos formal em todas as empresas e permite, através da distribuição das operações pelos recursos no tempo, maximizar ou minimizar uma determinada medida de desempenho. Baker & Trietsch [5], definem o Escalonamento de Operações como um processo de decisão que responde a duas questões:

- Como deverão ser distribuídos os recursos disponíveis pelas operações?
- Como deverão ser distribuídas no tempo as operações?

Assim, o Escalonamento de Operações compreende decisões de Afetação, respondendo à 1ª questão, e decisões de Calendarização/Sequenciação, respondendo à 2ª questão [3,4,5].

2.2.1. PROBLEMA DE AFETAÇÃO

O problema de afetação compreende decisões que precedem a calendarização ou sequenciamento e determina que recursos serão alocadas a que tarefas. É um problema de decisão, pois em ambientes onde existem vários recursos capazes de executar as mesmas tarefas (Máquinas Paralelas) é necessário selecionar que recursos vão executar que tarefas, de forma a otimizar uma determinada medida de desempenho. Um problema de afetação, com três tarefas e três máquinas, é representado na figura 2.

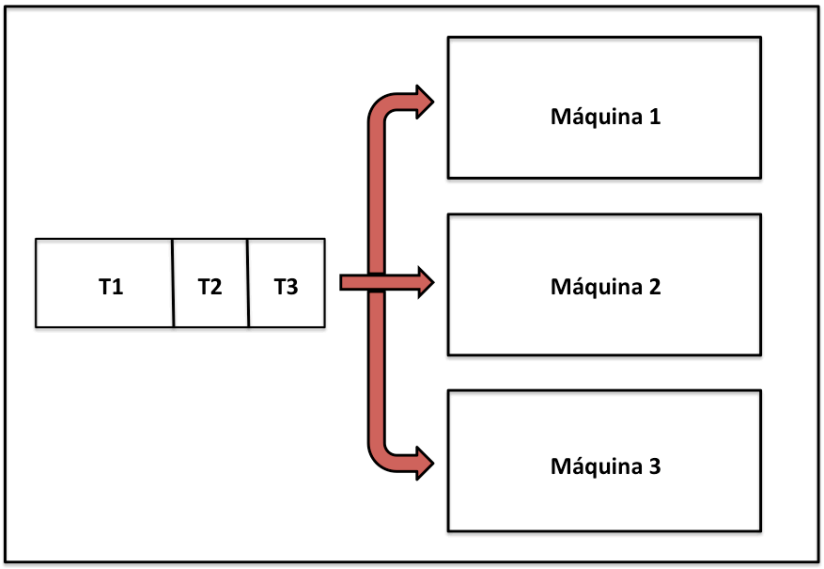

Figura 2 - Problema de Afectação

Em Blazewicz et al. [6], o problema de afetação é descrito por: "*Conhecidas n tarefas pertencentes a* $T = \{T_1, T_2, ..., T_n\}$, *m máquinas pertencentes a* $P = \{P_1, P_2, ..., P_m\}$ *e s recursos adicionais pertencentes a* $R = \{R_1, R_2, ..., R_s\}$, *como deverão as máquinas de P e recursos de R, ser afetados às tarefas de T por forma a maximizar/minimizar uma determinada função de desempenho?*" É habitual que uma tarefa apenas possa ser executada por uma máquina de cada vez e que cada máquina apenas possa executar uma tarefa de cada vez. Outra restrição bastante comum é a impossibilidade de interromper o processamento de uma tarefa uma vez iniciada.

Os problemas de afetação e os problemas de calendarização/sequenciamento, não são exclusivos da Teoria do Escalonamento. É comum abordar problemas de afetação, de máquinas, matérias-primas e mão-de-obra, através de modelos matemáticos que permitem encontrar soluções ótimas de uma forma eficiente [3,4,6,7].

2.2.2. PROBLEMA DE CALENDARIZAÇÃO/SEQUENCIAÇÃO

O problema de calendarização ou sequenciação compreende habitualmente decisões que sucedem a afetação. É um problema de decisão pois é necessário distribuir as operações no tempo, de forma a otimizar uma determinada medida de desempenho.

É importante separar a calendarização e a sequenciação, que embora determinem quando as operações devem ser executadas, não querem dizer a mesma coisa. Os problemas de calendarização determinam quando as tarefas devem ser executadas. Evidentemente que na presença de incerteza não é possível desenvolver um plano, que se tornaria obsoleto com qualquer variação nos tempos de execução. Os problemas de sequenciação determinam uma sequência de execução das tarefas, independentemente dos tempos de execução serem difusos. Em problemas determinísticos, onde os atributos do problema são conhecidos sem incerteza, a sequenciação é semelhante à calendarização, embora não permita especificar interrupções entre as tarefas, resultando sempre num plano de escalonamento sem atrasos. Um problema de sequenciação pode ser analisado na figura 3.

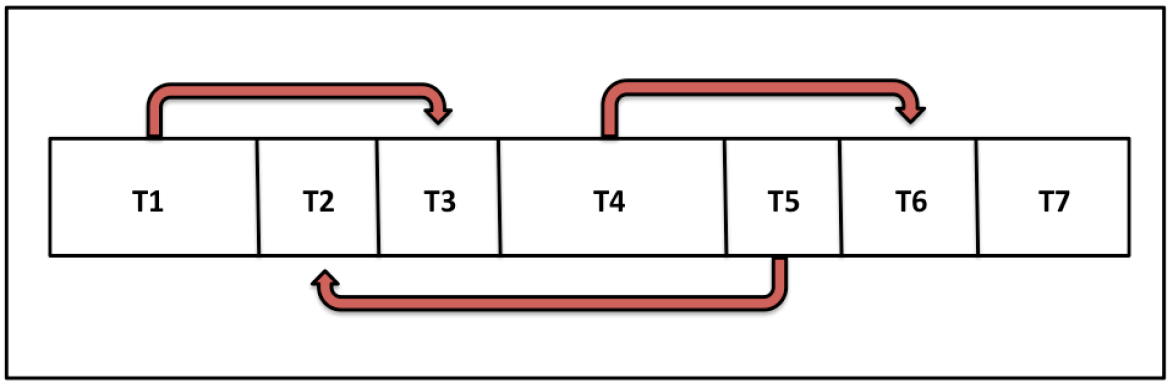

Figura 3 - Problema de Sequenciação

O problema de calendarização/sequenciação é descrito por: conhecidas n tarefas $T = \{T_1, T_2, ...,T_n\}$ e a sequência $S = \{S_1, S_2, ...,S_n\}$ e/ou o intervalo temporal [a,b]. Como devem as tarefas de T ser sequenciadas em S ou distribuídas em [a,b], para maximizar/minimizar uma determinada função de desempenho, sabendo que habitualmente apenas uma tarefa pode ocupar uma posição na sequência ou intervalo de tempo, numa dada máquina ?

Embora os problemas de calendarização ou sequenciamento aparentem serem simples, a sua natureza combinatória torna-os extremamente complexos. Habitualmente os problemas de calendarização/sequenciamento não podem ser resolvido de uma forma eficiente através de modelos matemáticos, ao contrário dos problemas de afetação [3,9].

2.3. DEFINIÇÕES FUNDAMENTAIS

Os principais desenvolvimentos na Teoria do Escalonamento apareceram em ambientes industriais. Assim, os recursos são denominados por máquinas, as atividades por tarefas, as partes de uma tarefas que apenas necessitam de uma máquina por operações e o ambiente produtivo é denominado por oficina [3,8]. É possível caracterizar cada elemento de um problema de escalonamento por:

- **Máquina** - Recurso automatizado ou humano necessário para executar uma tarefa, disponível de uma forma limitada. É caracterizado pela sua capacidade de executar as tarefas, que é presumivelmente constante.

- **Tarefas** - É uma atividade de trabalho localizada no tempo, para a qual é conhecida a sua data lançamento ou data de entrega. É caracterizado pela sua duração e pela intensidade com que consome recursos durante a execução.

- **Operações** - Em problemas onde as tarefas são executadas em partes, uma operação é parte de uma tarefa que apenas utiliza um recurso. Habitualmente apenas pode ser executada uma operação em cada momento.

- **Oficina** - É um local de fabrico onde estão implementados os recursos disponíveis. Os recursos podem estar dispostos de diversas formas, o que condiciona a forma como as tarefas vão circular através da oficina de fabrico.

Os recursos podem ainda ser renováveis, se ficarem novamente disponíveis depois de utilizados, ou consumíveis. Os recursos disjuntivos apenas podem executar uma tarefa de cada vez, enquanto recursos cumulativos podem executar várias simultaneamente [7,8].

2.3.1. ESTRUTURA DAS VARIÁVEIS

Existe uma estrutura definida para modelar um problema de escalonamento [7,8]. Sendo o número de máquinas representado por m, e o número de tarefas é representado por n, e sabendo que i representa as máquinas e j representa as tarefas é possível definir:

- p_{ij} - Representa o tempo de processamento da tarefa j na máquina i. Se for apenas representado por p_j é porque o tempo de processamento da tarefa não dependa da máquina onde vai ser processada, ou apenas pode ser processada numa máquina.

- r_j - Representa a data de lançamento da tarefa *j*, que não pode iniciar o seu processamento antes desse momento. Habitualmente representa o instante em que a ordem de fabrico é libertada na oficina para ser executada.

- d_j - Representa a data de entrega para a tarefa *j*, que se não for cumprida leva a organização a incorrer numa penalidade. Habitualmente indica uma data acordada com um cliente que ao não ser cumprida representa um custo de insatisfação.

- w_j - Representa o peso da tarefa *j* relativamente às outras tarefas do sistema, isto é, representa a importância ou prioridade relativa de uma tarefa numa medida de desempenho. Se não existir é porque todas as tarefas têm a mesma importância.

2.3.2. Implementação Industrial

Os problemas de escalonamento têm características semelhantes em ambientes onde as máquinas estão dispostas mesma forma. O escalonamento será dividido em problemas de:

- Máquina Única;
- Máquinas Paralelas;
- Linhas de Fabrico;
- Oficina de Fabrico.

Máquina Única - Os problemas de escalonamento em máquina única só compreendem decisões de calendarização/sequenciação. São muitas vezes utilizados para simplificar problemas mais complexos. Uma representação pode ser analisada na figura 4.

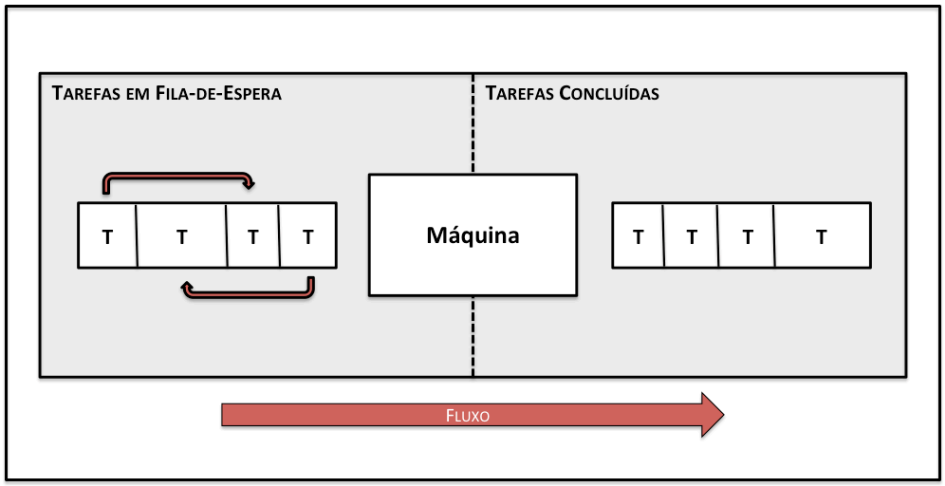

Figura 4 - Problema em Máquina Única

O Escalonamento das Operações em máquina única é um problema de sequenciação puro, isto é, um problema onde basta determinar uma sequência de tarefas para encontrar uma solução de escalonamento. Embora aparentemente simples é um problema de otimização combinatória, onde os trabalhos em fila-de-espera são permutados de maneira a otimizar uma determinada medida de desempenho. Baker & Trietsch [5], referem a importância deste problema, pois possibilita decompor problemas em ambientes de fabrico mais complexo em problemas mais simples, para encontrar soluções [4,7].

Máquinas Paralelas – O problema de escalonamento em máquinas paralelas, compreende decisões de afetação e de sequenciamento. Habitualmente os trabalhos são atribuídos às máquinas e só posteriormente sequenciados, isto é, a sequenciação é abordada através de problemas em máquina única. O problema pode ser analisado na figura 5.

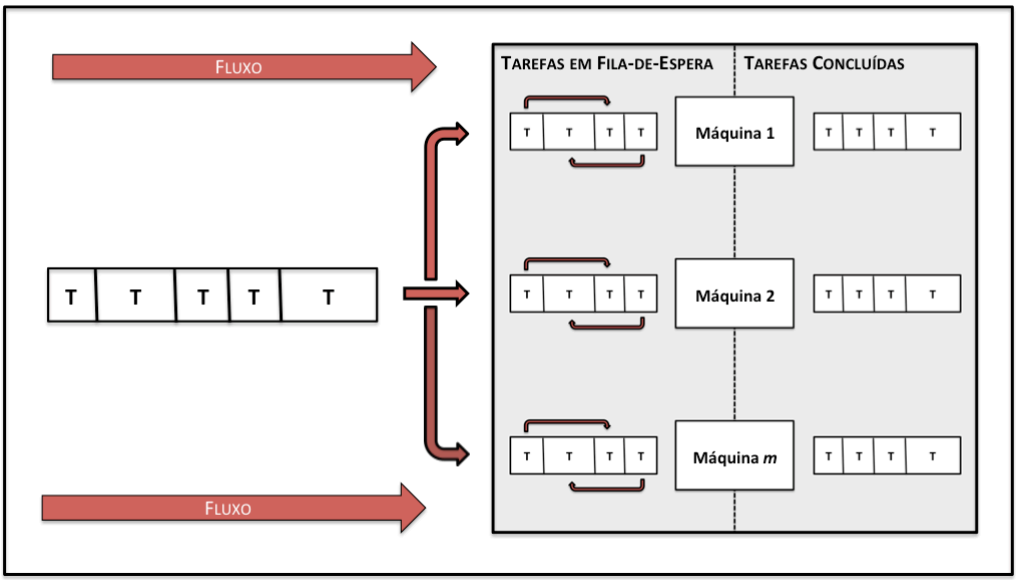

Figura 5 - Problema em Máquinas Paralelas

Os problemas em máquinas paralelas são problemas de escalonamento completos, isto é, problemas que compreendem decisões de afetação e de calendarização/sequenciação. Lopez, & Roubellat [7], caracterizam o problema como: "*Problema onde diversas máquinas são capazes de executar um determinado número de trabalhos, que, por sua vez, apenas necessitam de uma única máquina para serem executados*". Habitualmente as máquinas têm todas capacidades de processamento idênticas, embora existam problemas de máquinas paralelas com velocidades diferentes ou características completamente distintas. De forma idêntica ao problema em máquina única, os problemas de máquina paralela podem ser utilizados para abordar problemas em linhas de fabrico flexíveis através da decomposição dos problemas, em problemas distintos em máquinas paralelas.

É importante fazer distinção entre problemas em máquinas paralelas que permitem interrupções, isto é, problemas onde é possível interromper o processamento de uma tarefa depois de iniciado, e os problemas que não o permitem. Habitualmente os problemas de escalonamento em máquinas paralelas com interrupções são mais simples [4,5,7].

Linhas de Fabrico (*Flow-Shop*) - Os problemas em linhas de fabrico, habitualmente apenas compreendem decisões de calendarização/sequenciação. É um problema que ocorre periodicamente em situações industriais, onde várias tarefas têm que ser executadas numa série de máquinas na mesma sequência. O problema pode ser analisado na figura 6.

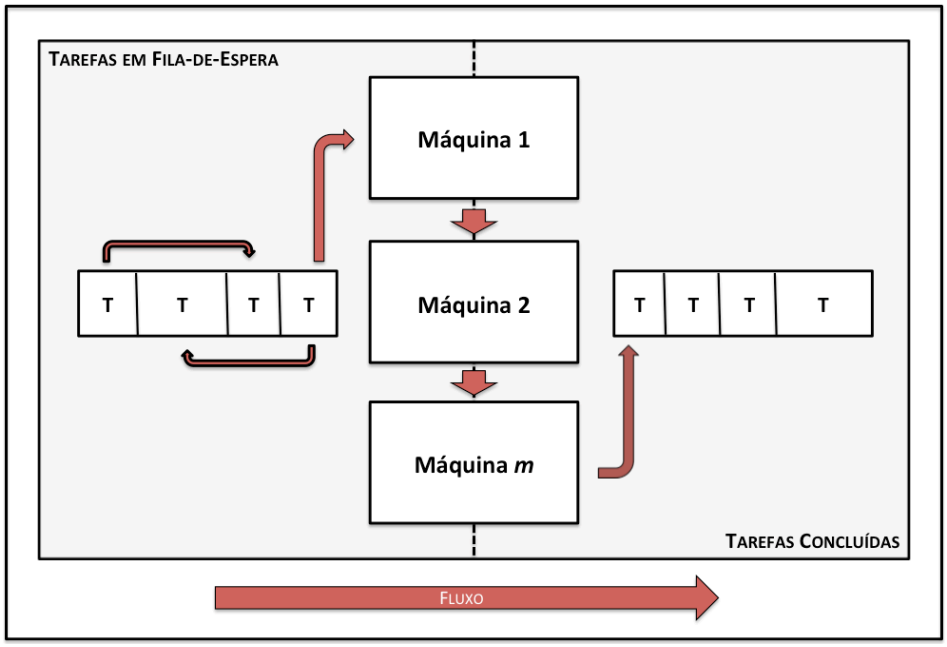

Figura 6 - Problema em Linhas de Fabrico

O Escalonamento de Operações em linhas de fabrico é tradicionalmente apenas constituído por decisões de sequenciação, isto é, são problemas onde basta determinar uma sequência de tarefas para encontrar uma solução. São problemas onde uma combinação de tarefas são processadas, pela mesma ordem, num determinado número de máquinas. Ao contrário dos problemas em máquina única ou máquinas paralelas, os problemas em linhas de fabrico decompõem as tarefas em *m* operações, que apenas utilizam um único recurso e devem ser executadas na sequência em que as máquinas estão dispostas.

É importante fazer distinção entre linhas de fabrico com armazéns intermédios, isto é, problemas onde existem tarefas em espera entre máquinas, e os problemas sem armazém intermédios. Os problemas em linhas de fabrico sem armazéns intermédios não permitem alterar a sequência de execução das tarefas entre máquinas [4,5,7].

Oficina de Fabrico (*Job-Shop*) - Os problemas em oficinas de fabrico compreendem decisões de afetação e calendarização/sequenciação. São problemas que ocorrem periodicamente, particularmente em unidades que executam elevados número de produtos em pequenas quantidades. O problema de escalonamento pode ser analisado na figura 7.

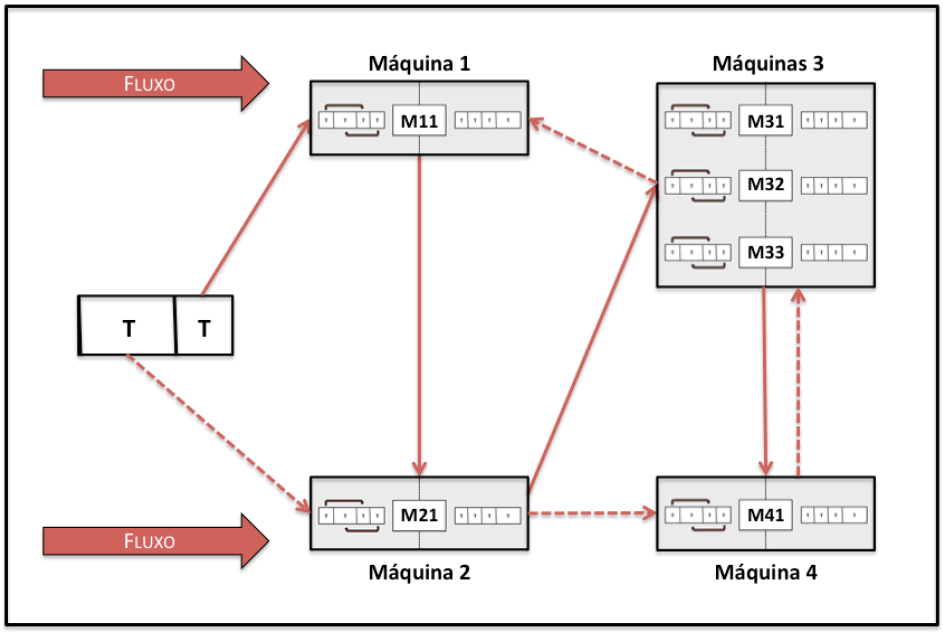

Figura 7 - Problema em Oficinas de Fabrico

O escalonamento em oficinas de fabrico é bastante complexo. Cada tarefa pode circular pelo sistema produtivo através de percursos diferentes, isto é, não existe uma sequência pré-determinada de máquinas para todas as tarefas. Embora, habitualmente, se considere que cada tarefa deverá visitar cada máquina uma única vez, existem casos onde as tarefas podem necessitar de duas operações na mesma máquina, ou mesmo, não necessitar de operações em determinadas máquinas. É também normal, considerar que apenas uma máquina poderá executar determinada operação, embora existam oficinas de fabrico com várias máquinas capazes de executar as mesmas operações.

Os problemas de escalonamento onde não existem limitações na maneira como as tarefas circulam através da oficina, são chamados de problemas em oficina aberta (*Open-Shop*), nestes problemas não existe uma sequência/ordem pré-determinada de operações, isto é, as operações podem ser executadas em qualquer sequência. São, tradicionalmente, problemas extremamente complexos, pois para além da afetação e calendarização/sequenciação das tarefas é necessário determinar uma sequência das operações. Os problemas de oficina aberta que permitam interrupções podem ser resolvidos facilmente, ao contrário dos problemas de oficinas de fabrico normais [4,5,7].

2.3.3. CRITÉRIOS DE OTIMIZAÇÃO

O correto Escalonamento de Operações procura maximizar/minimizar uma determinada medida de desempenho. Existem critérios de otimização internos, relacionadas com otimização dos recursos disponíveis e externos, relacionados com necessidades de clientes. Uma divisão mais habitual dos critérios de desempenho, considera:

- Os Tempo de Execução;
- A Pontualidade;
- A Produtividade;

Tempo de Execução – Os critérios de desempenho relacionados com o tempo de execução, medem a celeridade na execução das atividades, isto é, procuram rentabilizar o tempo de fabrico. Os critérios de otimização relacionados com o tempo de execução são:

- $\boldsymbol{C_j}$ - Representa a data de conclusão da tarefa *j*, isto é, o momento em que a tarefa é finalizada. Para tarefas constituídas por mais de uma operação, representa o momento em que a última operação é finalizada.

- $\boldsymbol{F_j}$ - Representa o tempo de execução da tarefa *j*, isto é, o tempo que vai demorar a percorrer o sistema produtivo. É calculado pela diferença entre a data de conclusão e a data de lançamento de uma determinada tarefa:

$$F_j = C_j - r_j \tag{1}$$

- $\boldsymbol{\overline{F}}$ - Representa o tempo médio de execução de uma ordem de fabrico, isto é, o tempo de execução médio de *n* tarefas. É calculado pela média das diferenças entre as datas de conclusão e de lançamento:

$$\overline{F} = \frac{1}{n}\sum_{j=1}^{n} F_j \tag{2}$$

- $\boldsymbol{C_{max}}$ – Representa o *makespan*, isto é, a data de conclusão de determinada ordem de fabrico. O *makespan* é equivalente ao valor máximo entre as data de conclusão de todas tarefas dessa ordem de fabrico:

$$C_{max} = max(C_j) \tag{3}$$

Pontualidade – Os critérios de desempenho relacionados com a pontualidade, medem a cumprimento dos prazos estipulados, isto é, procuram satisfazer as datas acordadas com os clientes. Os critérios de otimização relacionados com a pontualidade são:

- $\boldsymbol{L_j}$ – Representa o atraso ou antecipação de uma tarefa em relação a uma data definida. O atraso é calculado através da diferença entre a data de entrega e a data de conclusão de uma dada tarefa:

$$L_j = C_j - d_j \tag{4}$$

- $\boldsymbol{L_{max}}$ – Representa o atraso ou antecipação máximo de uma determinada ordem de fabrico, isto é, o máximo da diferença entre as datas de entrega e de conclusão, entre um dado número de tarefas:

$$L_{max} = max\left|L_j\right| \tag{5}$$

- $\boldsymbol{T_j}$ – Representa um atraso de uma tarefa em relação a uma data acordada, isto é, o atraso efetivo. Em T_j não são contabilizadas antecipações aos prazos estipulados, ao contrario do que acontece com L_j:

$$T_j = \max\left(L_j, 0\right) \tag{6}$$

- $\boldsymbol{T_{max}}$ – Representa o atraso máximo de uma determinada ordem de fabrico, isto é, o atraso efetivo máximo, que corresponde à tarefa com o maior atraso efetivo na ordem de fabrico:

$$T_{max} = max\left|T_j\right| \tag{7}$$

- $\boldsymbol{\overline{T}}$ - Representa o atraso médio de uma determinada ordem de fabrico, isto é, o atraso efetivo médio, que corresponde a média dos atrasos efetivos das tarefas na ordem de fabrico:

$$\overline{T} = \frac{1}{n}\sum_{j=1}^{n} T_j \tag{8}$$

- $\boldsymbol{E_j}$ – Representa uma antecipação de uma tarefa em relação a uma data acordada, isto é, um adiantamento. Em E_j não são contabilizados os atrasos aos prazos estipulados, ao contrário do de acontece com L_j:

$$E_j = \max(E_j, 0) \tag{9}$$

- $\boldsymbol{E_{max}}$ – Representa a antecipação máxima de uma determinada ordem de fabrico, isto é, o adiantamento máximo, que corresponde à tarefa com o maior adiantamento na ordem de fabrico:

$$E_{max} = max|E_j| \tag{10}$$

- $\boldsymbol{\overline{E}}$ - Representa a antecipação média de uma determinada ordem de fabrico, isto é, o adiantamento médio, que corresponde ao adiantamento médio das tarefas de uma ordem de fabrico:

$$\bar{E} = \frac{1}{n}\sum_{j=1}^{n} E_j \tag{11}$$

- $\boldsymbol{U_j}$ - Representa as tarefa que não cumprirem o seu prazo de entrega estipulada, isto é, uma tarefa que será concluída depois do prazo. Apenas são contabilizadas tarefas com um atraso efetivo:

$$U_J = \begin{cases} 1 \; se \; C_j > d_j \\ 0 \; se \; C_j \leq d_j \end{cases} \tag{12}$$

- $\boldsymbol{N_T}$ - Representa o número de tarefas que não cumprirem os seus prazos de entrega estipulados, isto é, o número de tarefas em atraso. Apenas são contabilizadas tarefas com um atraso efetivo:

$$N_T = \sum_{j=1}^{n} U_j \tag{13}$$

Produtividade – Os critérios desempenho relacionados com a produtividade medem a rentabilidade dos recursos produtivos, isto é, a eficiência do Escalonamento de Operações. Os critérios de otimização relacionados com a produtividade são:

- ***UR_i*** – Representa a taxa de utilização de uma máquina. É determinado através da relação entre o tempo de execução das operações numa máquina e o tempo total de execução de uma determinada ordem de fabrico:

$$UR_i = \frac{1}{C_{max}} \sum_{j=1}^{n} P_{ij} \times 100 \quad (14)$$

- ***UR*** – Representa a taxa de utilização de todo o sistema produtivo. É determinado através da relação entre o tempo de execução em todas as máquinas e o tempo máquina disponível:

$$UR = \frac{1}{C_{max} \times m} \sum_{i=1}^{m} \sum_{j=1}^{n} P_{ij} \times 100 \quad (15)$$

Embora habitualmente o Escalonamento de Operações procure otimizar uma única função de desempenho, em ambientes de fabrico reais é normal tentar otimizar vários critérios de otimização simultaneamente. Para se manterem competitivas num mercado extremamente volátil, as empresas têm que produzir rapidamente, cumprir os requisitos dos clientes e, ao mesmo tempo, rentabilizar os investimentos em recursos produtivos. Uma fábrica poderá tentar diminuir o seu tempo de execução e ao mesmo tempo aumentar a sua produtividade, o que poderá tornar os problemas extremamente difíceis, especialmente quando os critérios de otimização estão relacionados. Neste caso é necessário encontrar uma solução de compromisso entre os diferentes objectivos da empresa, com a ponderação de importância definida internamente [4,8,9].

2.4. REPRESENTAÇÃO DE GRAHAM

Como foi possível constatar os problemas de escalonamento podem ter características muito distintas. Problemas com ambientes de fabrico, restrições e critérios de otimização diferentes podem necessitar de ser abordados de maneiras distintas. Graham [10], introduziu um método de classificação dos problemas de escalonamento com três campos α|β|γ. Na representação de Graham α descreve o ambiente de fabrico, β as características ou restrições do processo produtivo e γ as medidas de desempenho do problema [4,6,8,9].

É de notar que existem outras formas de representar os problemas de escalonamento. Conway et al. [11] classifica os problemas de escalonamento em quatro campos *A*|*B*|*C*|*D*.

2.4.1. AMBIENTES DE PRODUÇÃO

O ambiente de produção, isto é, o número de máquinas disponíveis e a forma como estão dispostas é descrito no campo α da representação de Graham [4,6,8]. É possível identificar:

- ***1*** - Representa um problema de escalonamento em máquina única, isto é, quando todas as tarefas devem ser executadas utilizando apenas um recurso.

- $\boldsymbol{P_m}$ - Representa um problema de escalonamento em máquinas com velocidades idênticas, dispostas em paralelo, isto é, quando as tarefas podem ser executadas por uma de *m* máquinas com velocidades idênticas.

- $\boldsymbol{Q_m}$ - Representa um problema de escalonamento em máquinas com velocidades diferentes, dispostas em paralelo, isto é, quando as tarefas podem ser executadas por uma de *m* máquinas com velocidades diferentes.

- $\boldsymbol{R_m}$ - Representa um problema de escalonamento em máquinas com características diferentes, dispostas em paralelo, isto é, quando as tarefas podem ser executadas por uma de *m* máquinas com características diferentes.

- $\boldsymbol{F_m}$ - Representa um problema de escalonamento em máquinas dispostas em série, isto é, um problema onde as tarefas devem ser executadas sequencialmente em *m* máquinas, transitando da máquina *m* para a máquina $m+1$.

- $\boldsymbol{FF_c}$ – Representa um problema de escalonamento em centros de fabrico dispostos em série, isto é, um problema onde as tarefas devem ser executadas em cada um dos *c* centros de fabrico, por uma das *m* máquinas paralelas.

- $\boldsymbol{J_m}$ - Representa um problema de escalonamento numa oficina de fabrico, onde não existe uma sequência pré-determinada de máquinas, isto é, cada uma das tarefas não necessita de circular no sistema através da mesma percurso.

- $\boldsymbol{FJ_c}$ - Representa um problema de escalonamento em centros de fabrico, onde não há uma sequência de centros de fabrico idêntica para todas as tarefas. Em cada centro de fabrico existem *m* máquinas idênticas.

- ***O_m*** - Representa um problema de escalonamento com *m* máquinas e onde não existem restrições na forma como as tarefas se deslocam através do sistema. Neste problema as tarefas não necessitam de circular no sistema através da mesma rota.

2.4.2. CARACTERÍSTICAS DO PROCESSO

As características do processo de fabrico, isto é, as principais especificidades de fabrico são descritas no campo β da representação de Graham [4,6,8]. É possível identificar:

- ***r_j*** - Representa a existência de datas de lançamento, isto é, as tarefas apenas podem ser executadas depois da sua data de lançamento. Isto quer dizer que as tarefas não estão disponíveis para processamento ao mesmo tempo.

- ***s_{jk}*** - Representa a existência de tempos de *setpup* dependentes da sequência em que as tarefas são executadas. Isto quer dizer que existe um tempo de *setup* s_{jk} entre o processamento das tarefas *j* e *k*.

- ***prmp*** - Representa a possibilidade de interromper a execução das tarefas sem perder o trabalho concluído, isto é, uma tarefa pode ser suspensa e retomada a partir do ponto onde foi interrompida.

- ***prec*** - Representa a existência de relações de precedência na sequência de execução das tarefas, isto é, as tarefas apenas poderão ser executadas depois de todas as tarefas que a precedem serem concluídas.

- ***fmls*** - Representa a existência de famílias de tarefas. Isto quer dizer que dentro da mesma família, todas as tarefas, mesmo quando têm características diferentes, não necessitam de *setup* entre si.

- ***batch(b)*** - Representa a possibilidade das tarefas serem processadas em lotes, isto é, *b* tarefas podem ser executadas simultaneamente. O tempo de execução do lote é o da tarefa com o maior tempo de execução.

- ***brkdown*** - Representa a possibilidade das máquinas avariarem, isto é, nem todas as máquinas estão sempre disponíveis. Habitualmente é associado a uma distribuição estatística da fiabilidade das máquinas.

- ***prmu*** - Representa um problema onde não é possível fazer permutações na sequência de tarefas, isto é, depois de executadas na primeira máquina não é possível alterar a ordem de tarefas.

- ***nwt*** - Representa um Escalamento de Operações sem espera, isto é, uma tarefa apenas poderá ser executada quando tiver condições para circular através de todo o sistemas sem pausas entre operações.

- ***rcrc*** - Representa um problema de escalonamento com recirculação, isto é, um problema de escalonamento onde as tarefas poderão ter necessidade de visitar uma máquina mais do que uma vez.

2.4.3. MEDIDAS DE DESEMPENHO

As medidas de desempenho são descritas no campo γ da representação de Graham [4,6,8]. Entre as medidas de desempenho, as mais habituais são:

- ***Min*** C_{max} - Representa problemas de minimização do *makespan*, isto é, problemas onde se pretende reduzir o tempo de conclusão de uma ordem de fabrico.

- ***Min*** $\sum_{j=1}^{n} C_j$ - Representa problemas de minimização do tempo total de execução, isto é, minimizar o tempo de execução para todas as tarefas.

- ***Min*** $1/n \sum_{j=1}^{n} C_j$ - Representa problemas de minimização do tempo médio de execução, isto é, minimizar o tempo médio de execução de todas as tarefas.

- ***Min*** L_{max} - Representa problemas de minimização do atraso ou antecipação máxima, isto é, minimizar o maior atraso ou antecipação de uma ordem de fabrico.

- ***Min*** $\sum_{j=1}^{n} L_j$ - Representa problemas de minimização dos atrasos e antecipações, isto é, minimizar a soma de todos os atrasos e antecipações.

- ***Min*** $\sum_{j=1}^{n} w_j L_j$ - Representa problemas de minimização dos atrasos e antecipações pesadas, isto é, minimizar a soma ponderada dos atrasos e antecipações.

- ***Min*** $\sum_{j=1}^{n} U_j$ - Representa problemas de minimização do número de tarefas em atraso, isto é, minimizar o número de tarefas que não cumprirem a data de entrega.

2.5. TEORIA DA COMPLEXIDADE

O Escalonamento de Operações compreende problemas muito complexos. Blazewicz et al. [6], caracteriza a complexidade do algoritmo A, como: "*Em função da dimensão (n) da instância I do problema Π quantos ciclos elementares de processamento são necessários para resolver essa instância do problema através do algoritmo A*". Esta função de complexidade é independente da implementação do problema ou mesmo da capacidade de processamento do computador, pois independentemente da velocidade de processamento de cada ciclo elementar, será sempre necessário realizar o mesmo número de ciclos. A função de complexidade de um algoritmo é representada por $O(p(n))$, no caso de p ser polinomial então é um algoritmo de tempo polinomial, no caso de p não ser polinomial então é um algoritmo de tempo não-polinomial. Por exemplo, dois algoritmos com funções de complexidade n (Polinomial) e 2^n (Não-Polinomial), onde $n = 100$, necessitam de ciclos de processamento muito distintos, o primeiro necessita de 100 ciclos de processamento enquanto o segundo necessita de 1.28×10^{30} ciclos. Esta disparidade entre os números de ciclos apenas se vai acentuar com o aumento de n, pois num dos casos o número de ciclos vai crescer numa função polinomial (n) e noutro, numa função não-polinomial (2^n).

O Escalonamento de Operações compreende tanto problemas de decisão como problemas de otimização, os problemas de decisão são problemas com respostas binárias, isto é, sim ou não, os problemas de otimização têm respostas mais complexas que correspondem a uma solução que otimiza um critério de desempenho, como é possível analisar na figura 8.

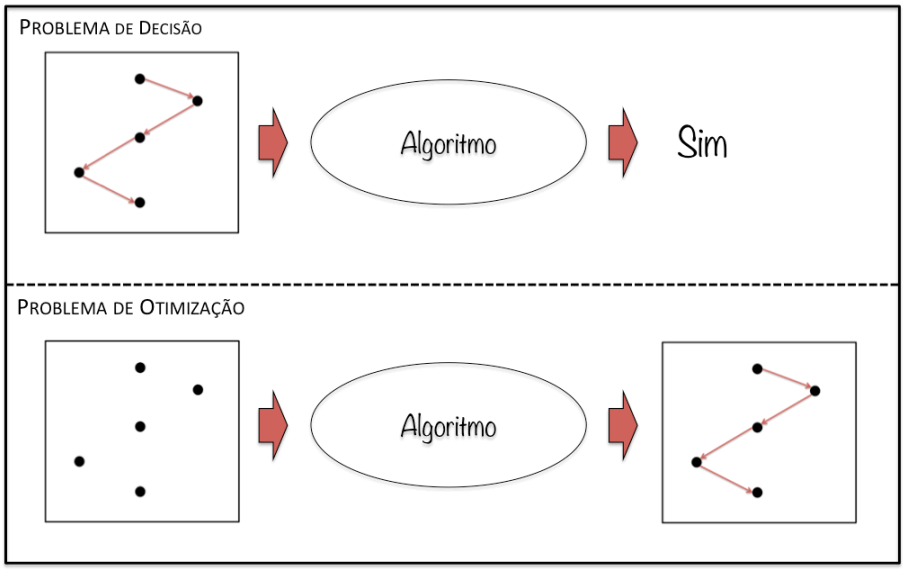

Figura 8 - Problema de Decisão e Problemas de Otimização

Em termos de complexidade os problemas de decisão são equivalentes aos problemas de otimização, isto é, se um problema de otimização poder ser resolvido eficientemente, o problema de decisão também pode. É então possível fazer uma distinção entre os problemas "*fáceis*", que podem ser resolvidos eficientemente, e os problemas "*difíceis*". Um problema de decisão da classe P pode ser resolvido em tempo polinomial por uma DTM (*Deterministic Turing Machine*), enquanto um problema de decisão da classe NP pode ser resolvido em tempo polinomial por uma NDTM (*Non-Deterministic Turing Machine*) [6]. Uma subclasses dos problemas *NP* são os problemas *NP-complete*. Os problemas de otimização NP podem ser transformados em problemas NP-complete em tempo polinomial, isto é, Π_1 é *NP-complete* se $\Pi_1 \epsilon\, NP$ se $\forall\, \Pi_2 \epsilon\ NP,\ \Pi_2 \propto \Pi_1$. Os problemas *NP-hard* são aqueles que são, pelo menos, tão difíceis como os problemas NP mais difíceis. Isto quer dizer que qualquer problema NP pode ser transformado num problema *NP-hard* em tempo polinomial, mas ao contrário dos problemas *NP-complete* os problema *NP-hard* não necessitam de pertencer aos problemas da classe NP [3,6,8].

2.5.1. PROBLEMAS COMBINATÓRIOS

O Escalonamento das Operações é muitas vezes de natureza combinatória. Os problemas de escalonamento que envolvem decisões de sequenciação são tipicamente combinatórios, e apresentam um tamanho máximo o qual possível resolver o problema. Por exemplo um problema de sequenciação em *n* tarefas numa única máquina, apresenta uma função de complexidade de *O(n!)*. Isto que dizer que um problema com *n = 4* têm 24 soluções, um problema com *n = 8* têm 40320 e um problema com *n = 12* têm 4.79×10^8, como é possível verificar um problema combinatório rapidamente se torna impraticável.

Os problemas de escalonamento combinatórios são por natureza problemas discretos e envolvem a determinação de uma ordem de execução de tarefas, isto é, cada solução é construída através da permutação da ordem de fabrico de um número finito de tarefas. Lopez & Roubellat [7], definem um problema combinatório, como: "*Sendo x as possíveis solução, f uma função de desempenho, qual é a solução s* ϵ x que otimiza f.*" Sendo *x* sempre um número finito de soluções definidas pelas restrições do problema de escalonamento. O problema de natureza combinatória pode ser representado por:

$$f(s^*) = \min_{s \in x} f(s) \tag{16}$$

Um exemplo de um problema de escalonamento combinatório pode ser analisado na figura 9, onde é necessário determinar uma ordem de execução de *n* tarefas. Tal como foi analisado, este problema rapidamente se torna intratável por métodos enumerativos.

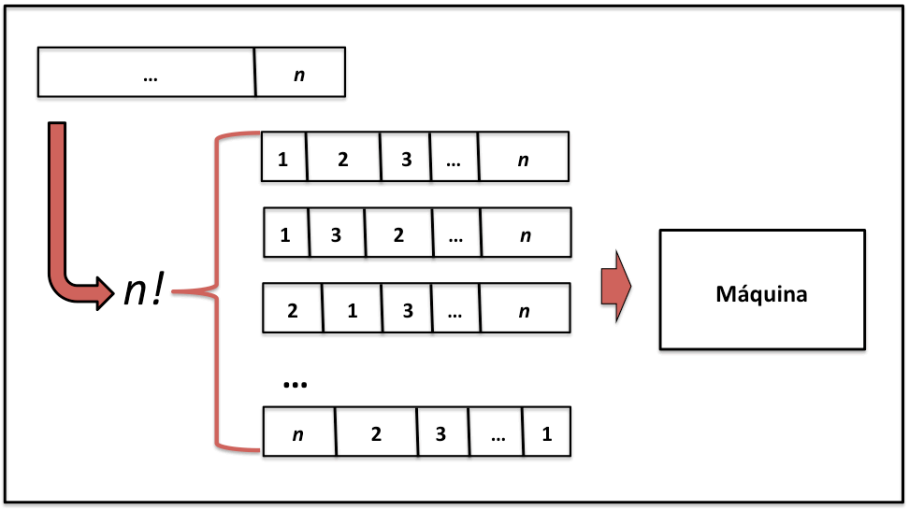

Figura 9 - Problema Combinatório

2.5.2. OUTRAS CARACTERÍSTICAS

Em Pinedo [4], é feita a distinção entre uma Sequência das Operações, o Escalonamento das Operações e as Políticas de Escalonamento. Enquanto uma Sequência das Operações apenas determina em que ordem as *n* tarefas devem ser executadas, o Escalonamento das Operações para além das decisões de sequenciamento deverá distribuir as operações pelos recursos. Por sua vez, as Políticas de Escalonamento resultam num procedimento reativo, isto é, perante um determinado estado do sistema implementam um ação pré-determinada. Existem problemas onde o Escalonamento de Operações apenas requer desenvolver uma Sequência de Operações, isto é, problemas onde determinar uma Sequência de Operações é o mesmo que determinar o Escalonamento das Operações.

O Escalonamento de Operações poderá ainda ter características que o demarcam dos outros. Estas características podem resultar de Políticas de Escalonamento, ou ser uma solução de escalonamento que resulta naturalmente da otimização de uma medida de desempenho. Uma solução de escalonamento poderá ser [4,9]:

- **Sem-Atrasos** - Um Escalonamento das Operações Sem-Atrasos é aquele onde nenhuma máquina pode ficar inativa quando existem operações por executar, isto é, um escalonamento onde não existe inatividade não forçada das máquinas.

- **Ativo** - Um Escalonamento das Operações Ativo é aquele em que não é possível desenvolver outra solução de escalonamento onde uma operação termine mais cedo sem causar um atraso numa outra operação.

- **Semi-Ativo** - Um Escalonamento de Operações Semi-Ativo é aquele em que não é possível desenvolver outra solução de escalonamento onde uma operação termine mais cedo sem alterar a sequência de execução.

2.6. TÉCNICAS DE OTIMIZAÇÃO

Os problemas de otimização podem ser abordados por técnicas diferentes, dependendo das suas características, um problema de otimização contínuo habitualmente não pode ser tratado da mesma forma que um problema de otimização combinatório. Enquanto os problemas de otimização contínuos poderão ser resolvido através de técnicas de otimização lineares ou não lineares, os problemas combinatórios, onde se incluem os problemas de escalonamento, recorrem tradicionalmente a técnicas enumerativas. Em problemas de otimização combinatórios existem técnicas exatas, aquelas que certificam soluções ótimas e técnicas aproximativas, ou heurísticas, que não garantem encontrar soluções ótimas. Em problemas onde apenas é necessário satisfazer as restrições, uma técnica é exata se certificar que todas as restrições são satisfeitas. Uma técnica de otimização é chamada de completa apenas se, independentemente do problema, for possível encontrar uma solução em tempo de computação reduzido. Na figura 10 é possível analisar as principais técnicas de otimização para problema de otimização combinatórios.

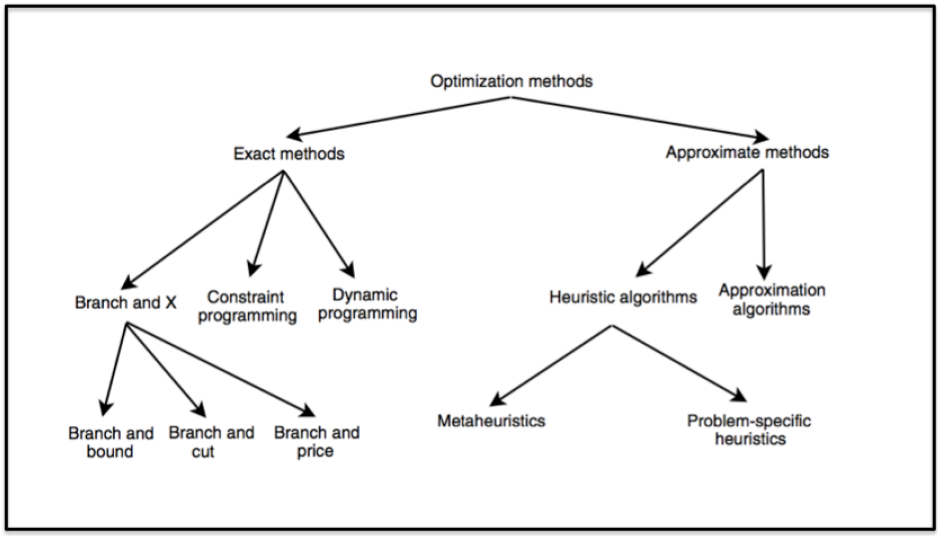

Figura 10 - Técnicas de Otimização (Adaptado de [12])

O Escalonamento de Operações inclui muitos problemas computacionalmente complexos, isto é, problemas que não podem ser resolvidos por técnicas de otimização exatas em reduzido tempo computacional. É normal abordar os problemas de escalonamento mais complexos através de técnicas de aproximação, mais ou menos evoluídas, que permitam obter um desempenho satisfatório em reduzido tempo de processamento [3,6,7,12].

2.6.1. TÉCNICAS EXATAS

São técnicas de otimização que garantem soluções ótimas independentemente do problema ou do tamanho da instância. Os problemas de escalonamento mais complexos habitualmente requerem a enumeração completa das soluções, para encontrar a solução ótima. O processo é tradicionalmente muito moroso, particularmente nos problemas de escalonamento que incluem decisão de sequenciamento, que rapidamente se tornam intratáveis através de técnicas enumerativas com o aumento da dimensão do problema. Entre as técnicas de otimização exatas é importante destacar:

- ***Branch and X*** – É uma famílias de métodos de otimização através de uma enumeração implícita que incluem o *Branch and Bound*, *Branch and Cut* e o *Branch and Price*. O problema de otimização é resolvido através de uma árvore de decisão onde cada ramo representa um subproblema e cada folha uma solução, permitindo reduzir o espaço de pesquisa apenas aos ramos e assim encontrar a solução ótima. O *branch* representa a divisão do problema de otimização em vários subproblemas, enquanto o *bound*, *cut* ou *price*, representam as técnicas de redução da pesquisa da árvore de decisão, isto é, as maneiras de encontrar uma solução ótima sem explorar todos os ramos da árvore [6,8,12].

- **Programação por Restrições** – É uma método de otimização que utiliza uma árvore de decisão e as relações entre as variáveis através de restrições. Sendo que cada variável pode adotar um número finito de valores, vão sendo aplicadas as restrições recursivamente, de maneira a encontrar soluções viáveis, isto é, é feita uma pesquisa nos valores possíveis das variáveis eliminado aqueles que não cumpram as restrições. O processo pode ser descrito como um processo de pesquisa e eliminação sistemática das soluções não viáveis. O problema pode procurar otimizar uma determinada medida de desempenho, ou simplesmente encontrar uma solução que cumpra todas as restrições [12].

- **Programação Dinâmica** - É um método de otimização de enumerativo que recursivamente divide o problema em subproblemas mais simples. É baseado no princípio de Bellman que diz: "*Uma subsolução de uma solução ótima é também ótima*". Isto quer dizer que a solução é construída recursivamente, através de uma sequência de soluções dos subproblemas, evitando assim enumerar soluções dos subproblemas não ótimas, que de acordo com o princípio de Bellman não vão resultar numa solução ótima. O problema pode ser abordado sequencialmente porque para qualquer subproblema as políticas ótimas são independentes das politicas dos subproblemas que o antecedem [6,8,12].

Os problemas de otimização contínuos podem ser resolvidos por métodos de otimização lineares ou não lineares, como por exemplo o *Simplex* ou Pontos Interiores. É importante notar que as técnicas de otimização exatas podem recorrer a técnicas de aproximação, ou serem utlizadas simultaneamente para desenvolver métodos híbridos [6,7,12].

2.6.2. TÉCNICAS APROXIMATIVAS

São técnicas de otimização que não garantem a solução ótima, isto é, métodos que equilibram a qualidade das soluções com o tempo de processamento necessário. Como é possível analisar na figura 10 as técnicas aproximativas podem ser divididas entre métodos heurísticos e métodos de aproximação, que garantem soluções de qualidade em tempos aceitáveis. Entre os métodos heurísticos é importante destacar:

- **Heurísticas Construtivas** - São métodos heurísticos de otimização que constroem uma solução de um problema, partindo de uma solução vazia a que vão sendo acrescentados elementos, isto é, são técnicas de otimização onde as soluções são construídas através de uma sequência de soluções de subproblemas. Habitualmente são chamadas de técnicas míopes, pois na construção de uma solução não tomam em consideração o impacto das decisões em fases posteriores [6,8,12].

- **Pesquisa Local** - São métodos heurísticos de otimização que pesquisam parte do espaço de soluções do problema, isto é, são técnicas de otimização que manipulam uma solução do problema para encontrar novas soluções com melhor desempenho. Habitualmente utilizam soluções aleatórias ou soluções de heurísticas construtivas como ponto de partida da pesquisa. Deslocam-se na direção das melhores soluções da vizinhança, podendo ficar bloqueadas em ótimos locais [6,8,12].

- **Meta-Heurísticas** - São uns dos principais métodos de otimização para problemas complexos. Pesquisam a totalidade do espaço de soluções do problema, isto é, são dotadas de mecanismos estocásticos que permitem obter maior diversidade na pesquisa do espaço de soluções e ultrapassar os ótimos locais. São habitualmente inspiradas em fenómenos observados na natureza e, podem ser divididas em Meta-Heurísticas de solução única ou Meta-Heurísticas de População [6,12].

Existem muitos problemas de escalonamento que podem ser abordados através de técnicas de otimização aproximativas mais simples, como Regras de Prioridade que determinam politicas de sequenciamento. Como foi referido anteriormente, os métodos aproximativos podem ser combinados com métodos exatos, em métodos de otimização híbridos [6,8,12].

2.7. ESCALONAMENTO DETERMINÍSTICO E ESTOCÁSTICO

Existe uma forte componente aleatória ou de incerteza no Escalonamento de Operações. Incerteza nos recursos disponíveis, com máquinas que avariam, ou novos requisitos dos clientes, podem tornar o Escalonamento de Operações obsoleto antes de implementado. Os tempos de execução, habitualmente, também não são conhecidos com precisão e por norma são descritos como tendo mais ou menos um determinado tempo de execução, isto é, um tempo de execução esperado, com determinada incerteza.

Os modelos de escalonamento podem aproximar esta componente aleatória ou incerteza, utilizando distribuições estatísticas nos tempos de execução ou probabilidades de avaria na disponibilidades das máquinas. O Escalonamento de Operações pode ser divido em:

- **Escalonamento Determinístico** - Onde se assume que as condições do problema são conhecidas. Serve para simplificar problemas reais que habitualmente incorporam sempre uma certa aleatoriedade, assumindo que todos os parâmetros do problema são conhecidos e podem ser descritos através de constantes [4].

- **Escalonamento Estocástico** - Onde se assume que as condições do problema não são conhecidas. É um problema de escalonamento onde os parâmetros do problema não são todos conhecidos e são tipicamente aproximados através de distribuições estatísticas, que permitam modelar a incerteza [4].

Um método para incorporar incerteza é o recurso a modelos fuzzy. Os modelos fuzzy são particularmente interessantes em problemas de satisfação de restrições, onde uma pequena variação dos parâmetros do problema pode tornar o problema mais fácil. Envolvem sempre compromisso entre os parâmetros de um problema e a qualidade da solução. Existem modelos fuzzy que apenas incorporam incerteza nas restrições e outros que permitem modelar incerteza em todos os parâmetros do problema. O modelo fuzzy procurar maximizar uma determinada medida de desempenho ao mesmo tempo que procura minimizar os desvios dos parâmetros, através de um modelo max/mim.

Em [13] é apresentado um modelo fuzzy que recorre ao SA (*Simulated Anneling*) para resolver um problema de escalonamento em máquinas paralelas. Os resultados, onde é atribuído um custo unitário ao desvio dos parâmetros do problema, permitiu encontrar soluções de qualidade perante um tempo de execução desconhecido ou incerto.

2.8. ESCALONAMENTO ESTÁTICO E DINÂMICO

Habitualmente o Escalonamento de Operações é abordado como um problema estático, isto é, como a distribuição de uma série pré-determinada de tarefas por recursos durante um período de tempo. Em ambientes industriais o Escalonamento de Operações estático é raro, tipicamente novas tarefas são lançadas durante a implementação de uma solução de escalonamento, tornando-a imediatamente obsoleta. Em tal caso, é necessário adaptar as soluções de escalonamento para acomodar as tarefas que vão sendo libertadas.

Os modelos de escalonamento podem aproximar esta componente aleatória ou incerteza, utilizando distribuições estatísticas nos tempos de execução ou probabilidades de avaria na disponibilidades das máquinas. O Escalonamento de Operações pode ser divido em:

- **Escalonamento Estático** – Onde se assume que as tarefas a ser executadas são conhecidas no início do problema e que mais nenhuma tarefa será lançadas antes da solução de escalonamento ser concluída.

- **Escalonamento Dinâmico** - Onde se assume que não são conhecidas todas as tarefas no início do problema, isto é, novas tarefas podem ser libertadas durante a implementação de uma dada solução de escalonamento.

Existe um componente estocástico num modelo de escalonamento dinâmico, no entanto um modelo estocástico poderá não ser dinâmico. Considera-se que um modelo dinâmico é sempre estocástico, pois lida com aleatoriedade no número de tarefas do problema, mas um modelo estocástico poderá não ser dinâmico se apenas lidar com aleatoriedade nos parâmetros de um determinado número de tarefas, conhecidas inicialmente.

O problema de escalonamento dinâmico é tratado através de modelos puramente reativos, que respondem ao lançamento de novas tarefas alterando a atual solução ou através de modelos preditivos, que antecipam o lançamento de novas tarefas. Habitualmente o escalonamento dinâmico reordena uma fila-de-espera das tarefas que ainda não foram executas (*Queueing Theory*), sempre que uma nova tarefa for libertada [4,14,15].

Outro conceito é o de escalonamento distribuído, onde um determinado número de tarefas necessitam de ser executadas em máquinas distribuídas em diferentes unidades produtivas, isto é, um problema de escalonamento colaborativo onde participam várias entidades. Em problemas de escalonamento distribuído é necessário recorrer a mecanismos que permitam às várias entidades coordenar as suas tarefas de maneira a otimizar uma medida de desempenho comum e ao mesmo tempo ter em conta as medidas de desempenho específicas de cada entidade do sistema [15,16]. Uma estrutura para um Sistema de Apoio à Decisão (DSS) é apresentado em [17], onde várias unidades produtivas partilham técnicas de escalonamento e coordenam as tarefas através de uma entidade centralizada, que monitoriza o desempenho e assiste na escolha das técnicas de escalonamento. O sistema proposto é capaz de aprender quais técnicas obtiveram os melhores resultados no passado, para melhor assistir às unidades produtivas que pertencem ao sistema.

2.9. PROBLEMAS DE ESCALONAMENTO TÍPICOS

Embora existem muitos problemas de escalonamento distintos, existem problemas, que pela sua frequente ocorrência em ambientes industriais ou pela sua simplicidade têm sido alvo de especial atenção. Será feita uma revisão dos problemas de escalonamento mais habituais nos quatro tipo de implantação mais frequentes:

- Máquina Única;
- Máquinas Paralelas;
- Linhas de Fabrico;
- Oficina de Fabrico.

É importante notar, que esta revisão da literatura não pretende ser particularmente exaustiva, apenas vai apresentar os problemas determinísticos mais comuns, para os quatro tipos de implementação mais frequentes, não incluído problemas em oficina aberta.

2.9.1. PROBLEMAS EM MÁQUINA ÚNICA

Os problemas de escalonamento em máquina única são interessantes pois podem ser utilizados para simplificar problemas em implementações mais complexas. São problemas de sequenciação puros, onde apenas é necessário determinar a sequência das tarefas para encontrar um solução de escalonamento. Embora os problemas de máquina única possam ser utilizados para simplificar problemas mais complexos não deixam de ser problemas de otimização combinatória, onde é necessário determinar uma sequência de execução. Os problemas em máquina única que vão ser abordados, são:

- Minimização do Tempos de Execução;
- Minimização do Atraso;
- Minimização do Número de Tarefas em Atraso.

Minimização dos Tempos de Execução - Os problemas de minimização do tempo total de execução, $1||\Sigma C_j$, são problemas de minimização do somatório dos tempos de execução de cada tarefa. O problema pode ser resolvido através do SPT (*Shortest Processing Time*) que ordena as tarefas por ordem não decrescente dos tempos de processamento. É possível analisar a comparação com LPT (*Longest Processing Time*) na figura 11. O método poderá ser adaptado para problemas de minimização do tempo de execução ponderado, $1||\Sigma w_j Cj$, no método WSPT (*Weighted Shortest Processing Time*) que ordena as tarefas por ordem não decrescente da razão entre os tempos de processamento e a ponderação [3,4,8].

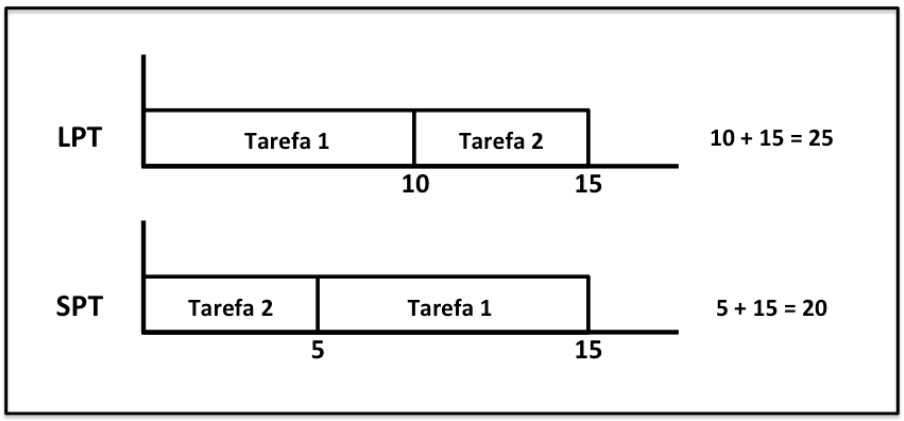

Figura 11 - Tempo Total de Execução (Adaptado de [3])

Minimização do Atraso - Os problemas de minimização do atraso máximo, $1||L_{max}$, são problemas de minimização do maior atraso entre as tarefas de uma determinada ordem de fabrico. O problema pode ser resolvido através do EDD (*Earliest Due Date*) que ordena as tarefas por ordem não decrescente das datas de entrega. O mesmo método pode ser utilizado em problemas de minimização do atraso efetivo máximo, $1||T_{max}$. Os problemas de minimização dos atrasos, $1||\Sigma L_j$, e problemas de minimização dos atrasos efetivos, $1||\Sigma T_j$, são *NP-hard* e não podem ser resolvidos otimamente pelo EDD, é necessário recorrer a técnicas de otimização exatas ou técnicas pseudo-polinomiais [3,4,8].

Minimização do Número de Tarefas em Atraso - Os problemas de minimização das tarefas em atraso, $1||\Sigma U_j$, procuram reduzir o número de tarefas que não vão cumprir as datas acordadas. É um problema particularmente importante num contexto industrial, sendo muitas vezes utilizado para medir o desempenho de sistemas produtivos. O EDD não é capaz de resolver estes problemas otimamente, embora existam instâncias onde é capaz de encontrar as soluções ótimas. É necessário recorrer a métodos *forward*, onde as tarefas são divididas entre aquelas que vão ser concluídas com atraso (A) e aquelas que vão ser concluídas antecipadamente (B). O método pode ser analisado na tabela 1 [3,4]:

Tabela 1 - Técnica de Otimização para $1||\Sigma U_j$

1º Passo:	*Colocar todos as tarefas em B ordenadas por EDD.*
2º Passo:	*Calcular as datas de conclusão das tarefas em B, se todas cumprirem as datas, terminar. Senão identificar os tarefas em atraso em B.*
3º Passo:	*Entre as tarefas em atraso identificar aquela com o maior tempo de processamento e move-la para A. Voltar para 2º Passo.*

2.9.2. PROBLEMAS EM MÁQUINA PARALELAS

Os problemas de escalonamento em máquinas paralelas podem compreender problemas de afetação e problemas de calendarização/sequenciação. É normal as máquinas terem todas características semelhantes, embora existam problemas com máquinas paralelas de velocidades ou características distintas. Os problemas de máquina paralela podem ser utilizados para simplificar os problemas em linhas de fabrico flexíveis, que podem ser decompostos em diversos problemas com máquinas paralelas.

Normalmente, os trabalhos são distribuídos pelas máquinas disponíveis e sequenciados posteriormente. Os problemas em máquina paralelas abordados, são:

- Minimização do Makespan sem Interrupções;
- Minimização do Makepsan com Interrupções;
- Minimização do Makespan em Máquinas Distintas.

Minimização do Makespan sem Interrupções - Os problemas de minimização do makespan em máquinas paralelas, $P_m||C_{max}$, são problemas de minimização do tempo de execução de uma ordem de fabrico. Estes problemas são *NP-hard* e podem ser abordados pela LPT (*Longest Processing Time*), que divide as tarefas por ordem não crescente do tempo de processamento. Isto vai permitir equilibrar os tempos em cada máquina com as tarefas com menor tempo de processamento. Os problemas com relações de precedência, $P_m|prec|C_{max}$, podem ser abordados pela LNS (*Largest Number of Successors*) [3,4,8].

Minimização do Makespan com Interrupções - Os problemas de minimização do makespan em máquinas paralelas com interrupções, $P_m|premp|C_{max}$, são problemas de minimização do tempo de execução de uma ordem de fabrico quando é possível interromper a execução de uma tarefas sem perder o trabalho concluído. Embora este problema também possa ser abordado pela LPT (*Longest Processing Time*), existe uma forma de resolver o problema otimamente. Sabendo que o makespan têm um limite inferior definido pela expressão 17, pode ser resolvido pelo método descrito na tabela 2 [3,4,8].

$$C_{max} \geq \left(p_{max}, \sum_{j=1}^{n} \frac{p_j}{m} \right) = C^*_{max} \qquad (17)$$

Tabela 2 - Técnica de Otimização para $P_m|premp|C_{max}$

1º Passo:	*Colocar todas as tarefas numa sequência aleatória numa máquina.*
2º Passo:	*Dividir a sequência de tarefas em intervalos do tipo $[0,C^*_{max}]$, $[C^*_{max},2C^*_{max}]$,..., $[(m-1)C^*_{max},mC^*_{max}]$.*
3º Passo:	*Distribuir os intervalos por cada uma das máquinas.*

Minimização do Makespan em Máquinas Distintas - Os problemas de minimização do makespan em máquinas distintas, $R_m||C_{max}$, são problemas de minimização do makespan quando cada uma das máquinas tem características diferentes, isto é, quando as tarefas têm tempos de processamento diferente em cada máquina. Estes problemas são NP-hard embora existam muitas heurísticas capazes de abordar o problema, como: MET (*Minimum Execution Time*), MCT (*Minimum Completion Time*) ou MOMCT (*Modified Ordered Minimum Completion Time*). A heurística MCT pode ser analisada na tabela 3 [8,18-20]:

Tabela 3 - Heurística MCT

1º Passo:	*Colocar todas as tarefas numa lista com uma sequência aleatória.*
2º Passo:	*Alocar a 1º tarefa à máquina onde têm a menor data de conclusão.*
3º Passo:	*Remover a tarefa alocada no 2º Passo da lista das tarefas.*
4º Passo:	*Atualizar as disponibilidades das máquinas.*
5º Passo:	*Repetir os passos 2, 3 e 4 até todas as tarefas terem sido alocadas .*

2.9.3. PROBLEMAS EM LINHAS DE FABRICO

Os problemas em linhas de fabrico apenas compreendem decisões de sequenciação. Isto quer dizer que apenas é necessário determinar uma sequência de tarefas para encontrar uma solução. Enquanto os problemas de máquina única ou máquinas paralelas, onde as tarefas são compostas por uma única operação, os problemas em linhas de fabrico cada tarefas é composta por *m* operações, que devem ser executadas na mesma sequência de máquinas. Existem linhas de fabrico com armazéns intermédios, onde podem existir tarefas em espera entre máquinas, e problemas sem armazém intermédios. O escalonamento em linhas de fabrico sem armazéns intermédios não permitem permutar a sequência das tarefas entre máquinas. Embora as linhas de fabrico tradicionais apenas incluírem decisões de sequenciamento, existem linhas de fabrico que incluem decisões de afetação: as linhas de fabrico flexíveis, que são constituídas por várias máquinas em paralelo, em cada posição. Apenas serão abordados dois problemas em máquinas paralelas:

- Minimização do Makespan;
- Minimização dos Tempos de Execução Ponderados.

Minimização do Makespan - Os problemas de minimização do makespan em linhas de fabrico, $F_m||C_{max}$, são problemas de minimização do tempo de execução de uma ordem de fabrico. O problema deverá ser abordado por técnicas distintas, dependendo do número de máquinas. Para duas máquinas, $F_2||C_{max}$, é possível encontrar a solução ótima, através do Algoritmo de Johnson, que pode ser analisado na tabela 4. O problema com mais de duas máquinas é mais complexo, no entanto, pode ser abordado através da heurística Slope, que é capaz de encontrar boas soluções. Esta heurística ordena as tarefas através de um índice calculado pelos tempos de processamento. É importante notar que esta heurística resulta numa solução sem permutações na sequência de execução entre máquinas [3,4,8].

Tabela 4 - Algoritmo de Johnson

1º Passo:	*Dividir as tarefas em dois Set. Colocar as tarefas onde $p_{1j} > p_{2j}$ no Set I e as tarefas onde $p_{1j} > p_{2j}$ no Set II. Tarefas onde $p_{1j} = p_{2j}$ podem ser colocadas em qualquer Set.*
2º Passo:	*Ordenar as tarefas do Set I através da SPT.*
3º Passo:	*Ordenar as tarefas do Set II através do LPT.*

Minimização dos Tempos de Execução Ponderados - O problema de minimização dos tempos de execução ponderados, $F_m||\Sigma w_j Cj$, é extremamente complexo embora existam restrições que toraem o problema mais simples. Em problemas em linhas de fabrico proporcionais e sequenciais, isto é, linhas de fabrico onde cada tarefas têm o mesmo tempo de processamento em cada uma das máquinas e onde não é possível alterar a ordem de execução das operações entre máquinas, $F_m|prmu,p_{ij}=p_j|\Sigma w_j Cj$, pode ser resolvido pelo WSPT-MCI (*Weighted Shortest Processing Time First with Minimum Cost Insertion)*, que poderá ser analisado em [4]. O WSPT-MCI pode também ser utilizado para problemas onde é possível alterar a sequência de execução das tarefas, $F_m|p_{ij}=p_j|\Sigma w_j Cj$ [3,4,8].

2.9.4. PROBLEMAS EM OFICINAS DE FABRICO

Os problemas em oficinas de fabrico deixam as tarefas circular pelo sistema através de percursos diferentes. Isto quer dizer que não existe uma sequência pré-determinada de máquinas, semelhante para todas as tarefas. Este facto torna os problemas em oficina de fabrico extremamente complexos e difíceis de abordar.

Existem problemas onde as tarefas podem necessitar de ser processadas várias vezes na mesma máquina, designados, problemas em oficinas de fabrico com recirculação, e problemas onde diversas máquinas podem executar uma operação, chamados de problemas em oficinas de fabrico flexíveis. Os problemas em oficinas abertas também podem ser analisados como um caso particular das oficinas de fabrico, onde não existem limitação nas deslocações das tarefas. Habitualmente os casos particulares tornam o problema mais complexo, os problemas analisados são em oficinas de fabrico tradicionais:

- Minimização do Makespan;
- Minimização do Tempos Execução Ponderados.

Minimização do Makespan - O problema de minimização do makespan em oficinas de fabrico, $J_m||C_{max}$, é um problema que procura minimizar o tempo de execução de uma ordem. O problema é extremamente complexo não sendo conhecido nenhum método para resolver o problema eficientemente, sendo habitualmente utilizada a heurística Shifting Bottleneck que poderá ser analisada na tabela 5. É baseado no princípio de *bottleneck*, que circula entre as máquinas do problema, que são abordadas individualmente [3,4,8].

Tabela 5 - Heurística Shifting Bottleneck

	M = Máquinas do problema.
	M_0 = Máquinas com tarefas sequenciadas.
1º Passo:	*Escolher o bottleneck resolvendo um problema $1|r_j|L_{max}$ para cada máquinas. A máquina bottleneck é aquela que tiver o atraso máximo.*
2º Passo:	*Sequenciar as tarefas na máquina bottleneck.*
3º Passo:	*Resequenciar as tarefas em todas as máquinas pertencentes a M_0.*
4º Passo:	*Repetir os passos 1, 2 e 3 até todas as máquinas pertencerem a M_0.*

Minimização dos Tempos de Execução Ponderados - O problema de minimização dos tempos de execução ponderados, $J_m||\Sigma w_jCj$, é outro problema complexo. O problema poderá ser abordado através de uma versão alterada da heurística *Shifting Bottleneck*, onde a máquina *bottleneck* é aquela que apresenta o maior atraso efetivo ponderado, $1|r_j|\Sigma w_jCj$. Esse problema poderá ser resolvido através da heurística ATC (*Apparent Tardiness Cost*) que ordena as tarefas de uma forma não crescente do custo de atraso ponderado [3,4,8].

2.10. CONCLUSÃO

Os conceitos fundamentais do problema de escalonamento foram abordadas neste capítulo, depois de um enquadramento histórico. Os desenvolvimentos da teoria do escalonamento foram muito importantes em contextos industriais, utilizando mesmo uma denominação semelhante à utilizada numa unidade produtiva. O problema do escalonamento procura fazer um uso eficiente dos recursos, sendo preponderante na sobrevivência das empresas. O problema de escalonamento pode ser dividido em problema de afectação, que distribui as tarefas pelos recursos, problemas de sequenciação/calendarização que distribuem as tarefas no tempo em dada máquina e problemas mistos onde as tarefas são distribuídas pelos recursos e posteriormente sequenciadas/calendarizadas.

Foram apresentados quatro tipos de implementação: máquina única, máquinas paralelas, linhas de fabrico e oficinas de fabrico. Esta classificação é importante pois os problemas de escalonamento na mesma implementação, habitualmente, apresentam características semelhantes. Foram também apresentados os principais critérios de otimização do escalonamento, que podem ser classificados em critérios de otimização relacionados com tempo de execução, pontualidade ou produtividade. Posteriormente foram descritas as principais restrições de fabrico dos problemas de escalonamento e a notação de Graham.

Analisaram-se as principais dificuldades inerentes ao problema de escalonamento, com a análise da teoria da complexidade e uma descrição dos problemas de natureza combinatória. A natureza combinatória da maioria dos problemas de escalonamento limita as técnicas de otimização que podem ser utilizadas para resolver os problemas de escalonamento eficientemente. Assim, foi feita uma análise das principais técnicas de otimização, que podem ser divididas em técnicas exatas, tradicionalmente morosas mas que encontram sempre a solução ótima e técnicas de aproximação, que equilibram o tempo computacional com a qualidade da solução. Posteriormente foram analisadas as diferenças entre escalonamento determinístico e estocástico e estático e dinâmico.

O capítulo terminou com uma revisão da literatura, onde foram apresentadas formas de abordar vários problemas de escalonamento. Foram apresentados problemas em máquina única, máquinas paralelas, linhas de fabrico e oficinas de fabrico, que podem ser abordados por heurísticas, com bons resultados e sem necessitar de muito tempo computacional.

3. META-HEURÍSTICAS

Neste capítulo serão apresentados os principais conceitos e o funcionamento das meta-heurísticas. Serão também abordadas as técnicas de pesquisa local, que apresentam características semelhantes às utilizadas em muitas das principais meta-heurísticas. Serão ainda descritas as diferenças entre ótimos locais e ótimos globais e aborda a dicotomia entre intensificação e diversificação. Posteriormente serão analisadas as principais meta-heurísticas e examinadas as técnicas de parametrização mais usuais. Finalmente será realizada uma descrição das híper-heurísticas, que têm sido alvo de enorme interesse.

3.1. ENQUADRAMENTO HISTÓRICO

O conceito de técnica de otimização aproximativa é introduzido na primeira metade do século XX, com o desenvolvimento de técnicas de pesquisa local durante os anos 50. Já nos anos 70, são apresentadas as primeira heurísticas construtivas para problemas combinatórios que não podiam ser resolvidos eficientemente através das técnicas exatas. O desenvolvimento da teoria da complexidade, proporcionou uma procura por técnicas de otimização para problemas complexos, que nas décadas posteriores levou ao desenvolvimentos de muitas meta-heurísticas, embora existam meta-heurísticas anteriores aos desenvolvimentos da teoria da complexidade. O próprio termo "meta-heurística" apenas data de 1986 [21], anteriormente eram chamadas "heurísticas modernas" [22].

O desenvolvimento de meta-heurísticas foi particularmente intenso durante os últimos 30 anos, no entanto uma das principais meta-heurísticas data dos anos 60/70. Os GA (*Genetic Algorithms*) foram desenvolvidos por Holland [23], inspirados na Teoria da Evolução de Charles Darwin. Nos anos 80 várias meta-heurísticas são propostas: Na primeira metade da década é proposto o SA (*Simulated Annealing*), inspirado no tratamento térmico de metais, e em 1986 é proposto o TS (*Tabu Search*), inspirado no conceito da memória. São, ambas, técnicas baseadas em pesquisa local, com uma componente estocástica. Em 1986 é proposto o AIS (*Artificial Immune Systems*) e no final da década o GRASP (*Greedy Adaptive Search Procedure*). Nos anos 90 são propostas várias técnicas de optimização evolutiva, como o ACO (*Ant Colony Optimization*), inspirada no comportamento social das formigas, o PSO (*Particle Swarm Optimization*), inspirado no comportamento de bandos de pássaros, e o CA (*Cultural Algorithms*), inspirado na evolução cultural. Em 2001 é proposto o HS (*Harmony Search*), inspirado na improvisação musical, e em 2005 é proposto o ABC (*Artificial Bee Colony*), inspirada em colónias de abelhas. É possível analisar na figura 12, as meta-heurísticas desenvolvidas de 1947 até 1996 [12,24].

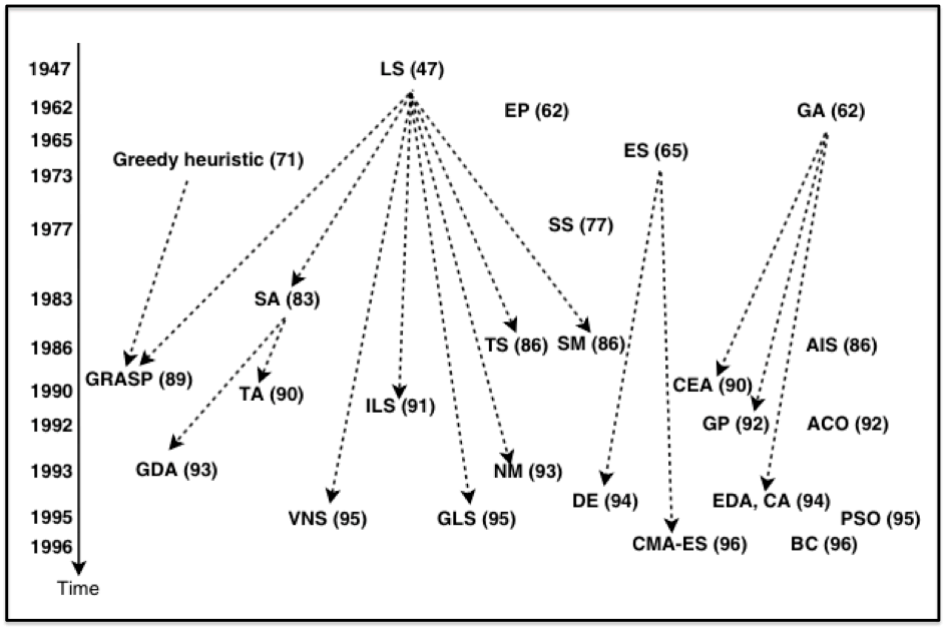

Figura 12 - Evolução das Meta-Heurísticas [12]

Atualmente as meta-heurísticas são uma das principais técnicas para abordar problemas computacionalmente complexos, como os problemas de otimização combinatória, onde demostraram ser as técnicas de aproximação com melhor desempenho [25,26]. Podem ser definidas como procedimentos iterativos que orientam uma heurística subordinada na exploração da totalidade do espaço de soluções, inspirados na natureza [12,27].

3.2. PESQUISA LOCAL

Os métodos de pesquisa local (*Local Search* - LS) são uma técnica de otimização aproximativa que fazem pesquisas repetidas numa parte limitada do espaço de soluções. Partindo de uma solução, habitualmente aleatória, é feita uma pesquisa na vizinhança por soluções com melhor desempenho, para substituir a atual solução. O processo vai repetir-se até não ser possível encontrar uma solução com melhor desempenho na vizinhança da atual solução. O conceito de vizinhança é fundamental às técnicas de pesquisa local, no entanto para o analisar é necessário compreender o conceito de codificação, isto é, a forma de representação das soluções. Esta representação das soluções deverá ser [28]:

- **Completa** - A representação das soluções do problemas deverá permitir representar todas as soluções do problema.
- **Conectável** - A representação das soluções do problemas deverá permitir encontrar todas as soluções do problema.
- **Eficiente** - A representação das soluções do problemas deverá permitir manipular as soluções do problema eficientemente.

O problema de sequenciamento de três tarefas (T1, T2 e T3) poderá ser representado através de um vector onde as tarefas são colocadas na ordem de execução. Por exemplo [2;3;1] que corresponde a uma soluções onde a tarefa 2 é a primeira a ser processada, depois a tarefa 3 e finalmente a tarefa 1. O mesmo problema poderá ser representado através de um vetor que indica as posições das tarefa. Por exemplo [2;3;1] que corresponde a uma solução onde a tarefa 3 é a primeira, depois a tarefa 1 e finalmente a tarefa 2. Existem formas de representação mais apropriadas para determinados problemas, os problemas de escalonamento habitualmente utilizam as representações demonstradas.

Uma vez descrito o processo de codificação, é possível descrever o conceito de vizinhança. Soluções são vizinhas, apenas se estiverem separadas por um movimento elementar, isto é, soluções vizinhas podem ser manipuladas de maneira a transformar uma na outra. Uma política de vizinhança é uma forma preestabelecida de manipular as soluções, por exemplo troca entre pares, na solução [2;3;1] os vizinhos seriam [3;2;1], [1;3;2] e [2;1;3], outra política de vizinhança poderá mover uma das tarefa, onde os vizinhos seriam [3;2;1], [3;1;2], [2;1;3] e [1;2;3]. Existe um número particularmente ilimitado de politicas de vizinhança e deverá ser escolhida uma que não resulte num número excessivo de vizinhos.

Os métodos de pesquisa local (ou de vizinhança) utilizam o conceito de vizinhança para pesquisar por soluções com melhor desempenho, dependendo da escolha de uma estrutura de vizinhança apropriado. Estes métodos fazem movimentar a pesquisa para as melhores soluções na vizinhança da solução atual, isto vai se repetir até não ser possível melhorar uma determinada solução. Isto é, no problema de minimização de $f(x)$ com um número finito de soluções x, partindo de solução inicial, tradicionalmente aleatória, $x_1 \in x$, na iteração n é escolhida uma nova solução x_{n+1} na vizinhança $v(x_n)$, sendo que $f(x_{n+1}) < f(x_n)$. O método pode ser analisado na tabela 6 [12,29,30].

Tabela 6 - Técnica de Pesquisa Local

1º Passo:	*Escolher uma soluções inicial* $x_n = x_1 \in x$.
2º Passo:	*Encontrar a melhor solução na vizinhança da atual solução* $v(x_n)$.
3º Passo:	*Se* $f(x_{n+1}) < f(x_n)$ *então* $x_n = x_{n+1}$ *e voltar ao 2º passo.*

Como é possível analisar, apenas parte da vizinhança é considerada para substituir a atual solução, isto é, apenas soluções com um desempenho superior ao da atual solução podem ser consideradas como próximas soluções. Isso torna as técnicas de pesquisa de vizinhança bastante eficientes, pois além de apenas explorar uma parte do espaço de soluções, só consideram os movimentos que melhoram a solução. O desempenho desta técnica poderá ser limitado por estas características, pois apenas vão explorar uma parte das soluções. É possível analisar na figura 13 as zona de movimentos admissíveis, uma representação do funcionamento da técnica e as suas principais limitações.

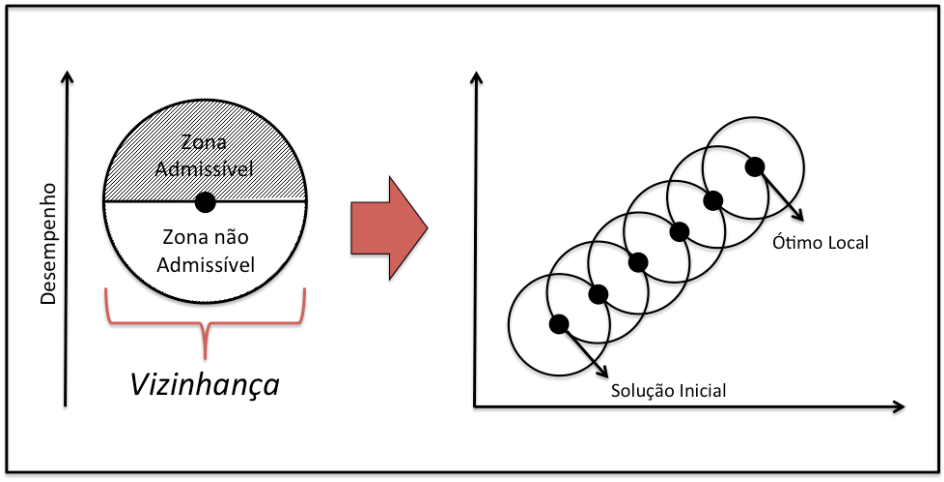

Figura 13 - Funcionamento da Pesquisa de Vizinhança [adaptado de 13]

3.2.1. EXEMPLO DA PESQUISA LOCAL

Para ilustrar o funcionamento da técnica de pesquisa local (ou de vizinhança) será utilizado um problema de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, com seis tarefas. Os dados do problema ilustrativo pode ser analisados na tabela 7.

Tabela 7 - Problema Ilustrativo

	1	2	3	4	5	6
p_j	6	2	5	4	3	6
d_j	9	12	15	11	17	18
w_j	1	2	3	3	2	3

Partindo da solução 1 2 3 4 5 6 com um resultado de 48 o problema vai ser abordado pela técnica de pesquisa local. A simulação da LS pode ser analisada na tabela 8. No exemplo foram encontradas duas soluções com melhor desempenho, 1 2 4 3 5 6 com um atraso efetivo ponderado de 39 e posteriormente 1 4 2 3 5 6 com 36, que é um ótimo local.

Tabela 8 - Exemplo do LS

	Solução	***f(x)***
1ª Iteração	2 1 3 4 5 6	48
	1 3 2 4 5 6	50
	1 2 4 3 5 6	**39**
	1 2 3 5 4 6	51
	1 2 3 4 6 5	51
2ª Iteração	2 1 4 3 5 6	39
	1 4 2 3 5 6	**36**
	1 2 3 4 5 6	48
	1 2 4 5 3 6	42
	1 2 4 3 6 5	42
3ª Iteração	4 1 2 3 5 6	37
	1 2 4 3 5 6	39
	1 4 3 2 5 6	40
	1 4 2 5 3 6	39
	1 4 2 3 6 5	39

3.3. ÓTIMOS LOCAIS E ÓTIMOS GLOBAIS

O LS desloca as soluções para aquelas com melhores desempenho, nas imediações da atual solução, num procedimento iterativo. É importante notar, que essa pesquisa é sempre feita numa determinada vizinhança, isto é, apenas são consideradas como próximas soluções, aquelas que se encontram próximas da atual solução e apenas se tiverem um desempenho superior ao da solução atual. Isto faz com que as técnicas de LS se desloquem sempre para um ótimo local, como apresentado na figura 13. Assim, x_1 é um ótimo local se:

$$f(x_1) \geq f(x), \qquad \forall\, x \in v(x_1) \tag{18}$$

As soluções que tenham o melhor desempenho entre todas as soluções possíveis do problema, são chamadas de ótimos globais. Isto quer dizer que uma técnica de LS apenas pode encontrar a solução ótima do problema se estiver nas vizinhança da solução inicial. No caso de existirem vários ótimos nas vizinhança da solução inicial, as técnicas de LS deslocam-se para aquelas que melhorem o desempenho mais rapidamente. Assim, x_1 apenas é um ótimo global se:

$$f(x_1) \geq f(x), \qquad \forall\, x \tag{19}$$

O principal problema das técnicas de LS é o facto de não ser possível evitar os ótimos locais, como é possível analisar na figura 14 e no exemplo da LS.

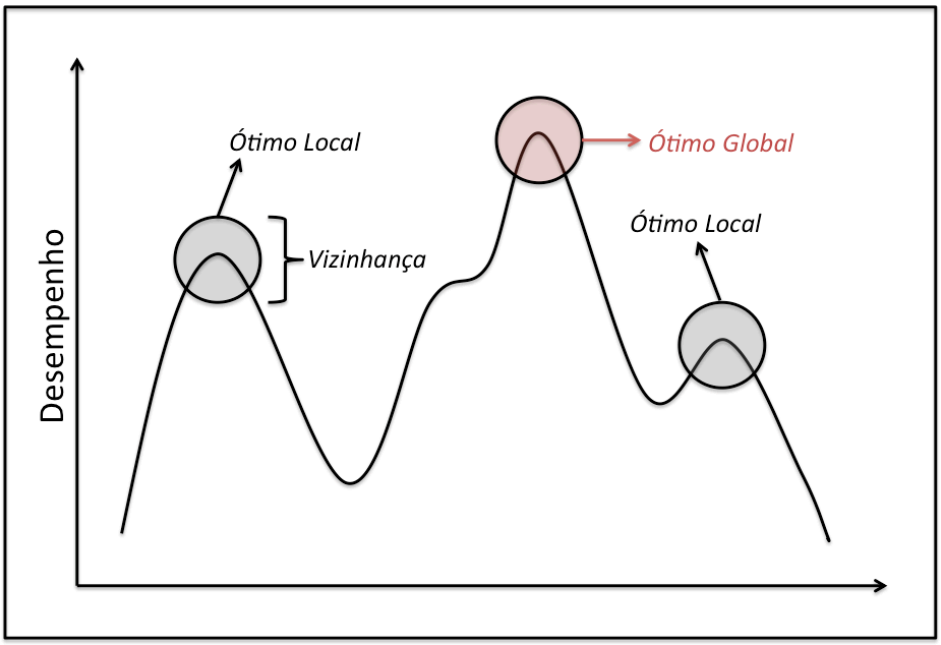

Figura 14 - Ótimo Locais e Ótimos Globais

3.4. INTENSIDADE E DIVERSIDADE

O conceito de meta-heurísticas é inspirados nos princípios de intensificação e diversificação. Intensidade, representa uma exploração minuciosa de parte, habitualmente uma parte com características promissoras, do espaço de soluções, enquanto, diversidade, representa uma exploração da totalidade do espaço de soluções, de maneira a evitar os ótimos locais. Esta dicotomia de conceitos representa uma das principais mais valias das meta-heurísticas, que devem efetuar uma pesquisa orientada, mantendo uma perspectiva da totalidade do espaço de soluções, isto é, um equilíbrio entre uma determinada vizinhança e uma componente estocástica na pesquisa do espaço de soluções. Enquanto muita intensidade poderá resultar numa confluência para um ótimo local, muita diversidade poderá resulta numa pesquisa aleatória, isto é, uma pesquisa não orientada.

O equilíbrio entre intensidade e diversidade representa a relação entre eficiência e eficácia da técnica de otimização. Uma pesquisa muito intensa vai confluir muito rapidamente num ótimo local, encontrando uma solução muito depressa, enquanto uma pesquisa com muita diversidade deverá ser morosa. Os métodos de LS têm muita intensidade e vão sempre mover-se para as melhores soluções numa determinada vizinhança, isto é, apenas vão explorar uma pequena parte das soluções; por sua vez uma pesquisa completamente aleatória vai movimentar-se através das soluções sem confluir para um único ponto. Embora este equilíbrio possa ser controlado através dos parâmetros das meta-heurísticas, existem técnicas, que pela suas características são mais intensas ou têm mais diversidade, como é possível analisar na figura 15 [7,12,28].

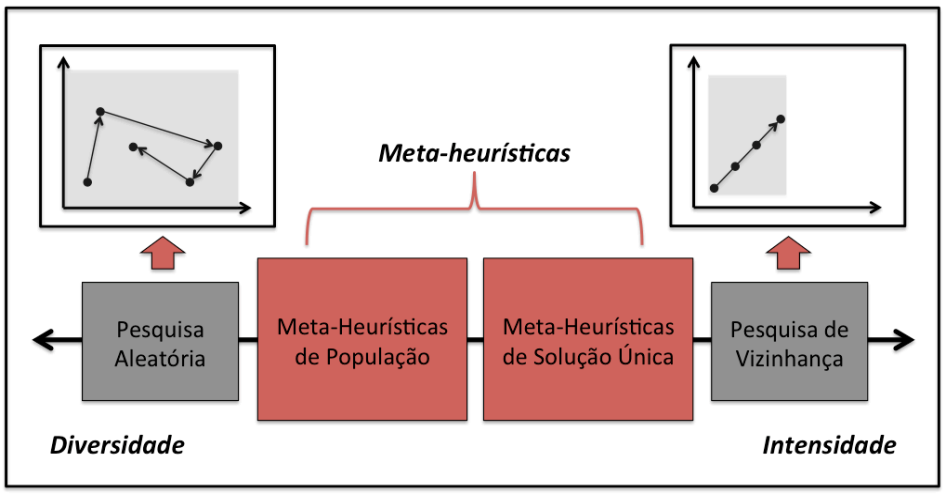

Figura 15 - Intensidade e Diversidade [adaptado de 12]

3.5. META-HEURÍSTICAS DE SOLUÇÃO ÚNICA E DE POPULAÇÃO

Existem duas classes distintas de meta-heurísticas, como foi possível analisar na figura 15. Existem meta-heurísticas de solução única e meta-heurísticas de população, isto é, as meta-heurísticas são divididas de acordo com o número de soluções que utilizam. Essa distinção também poderá ser analisada na figura 16, onde é representada a inspiração no LS das meta-heurísticas de solução única e descendência do GA das meta-heurísticas de população. Habitualmente as meta-heurísticas de solução única têm mais intensidade que as meta-heurísticas de população, que ao utilizarem várias soluções simultaneamente vão explorar mais partes do espaço de soluções, o que as torna mais morosas [12,22,25,31].

O conceito das meta-heurísticas de solução única é baseado na LS, acrescentando uma componente estocástica para ultrapassar os ótimos locais. Partindo de uma solução, estas técnicas vão fazer deslocar a pesquisa para as melhores soluções numa vizinhança como acontece na LS, mas com uma componente estocástica, isto é, permitido afastar-se do caminho de pesquisa pré-determinado e deslocar-se para além dos ótimos locais. A literatura identifica várias meta-heurísticas de solução única, as mais utilizadas são [12]:

- *Greedy Adaptive Search Procedure* (GRASP);
- *Iterative Local Search* (ILS);
- *Simulated Annealing* (SA);
- *Tabu Search* (TS);
- *Variable Neighborhood Search* (VNS).

O conceito das meta-heurísticas de população utilizam as características da população para controlar a pesquisa, manipulando varias soluções em simultâneo. Partindo de uma população, isto é, utilizando múltiplas soluções iniciais, estas técnicas vão evoluir a atual população para otimizar um dado critério de desempenho. Ao utilizar múltiplas soluções iniciais, realizam uma exploração mais ampla do espaço de soluções. A literatura identifica várias muitas meta-heurísticas de população, as mais utilizadas são [12]:

- *Ant Colony Optimization* (ACO);
- *Artificial Bee Colony* (ABC);
- *Cultural Algorithms* (CA);
- *Genetic Algorithms* (GA);
- *Particle Swarm Optimization* (PSO).

3.6. META-HEURÍSTICAS E ALGORITMOS

Os problemas que são demasiados complexos para serem resolvidos otimamente através de métodos exatos, isto é, onde é demasiado moroso enumerar todas as soluções, podem ser resolvidos através de meta-heurísticas. Osman & Kelly [26] apresentam as meta-heurísticas como o método de aproximação com melhor desempenho para problemas onde heurísticas construtivas demostram ser ineficientes. No contexto do escalonamento Xhafa & Abraham [25] afirmam que "*As meta-heurísticas são as técnicas predominantes, para abordar problemas de escalonamento complexos*", isto é, a natureza combinatória dos problemas de escalonamento, torna-a-os impossíveis de serem resolvidos eficientemente através de técnicas de otimização exatas, sendo habitual recorrer às meta-heurísticas.

Uma meta-heurísticas é um mecanismo que orienta uma heurísticas subordinada, na exploração do espaço de soluções para encontrar soluções próximas da ótima. Estas técnicas, ao contrário dos métodos de pesquisa local, que apenas analisam entre as soluções que pertencem a uma determinada vizinhança, são dotadas de mecanismos que permitem deslocar as soluções para além dos ótimos locais, tornando-as particularmente interessantes em problemas de otimização combinatórios. Como método de aproximação, as meta-heurísticas, não garantem encontrar soluções ótimas, mas habitualmente apresentam um desempenho superior às técnicas de aproximação menos evoluídas. Podem ser descritas como métodos de compromisso entre eficácia e eficiência das soluções [28].

O desenvolvimento de meta-heurísticas foi particularmente intenso durante as últimas 3 décadas e atualmente existe um enorme número de meta-heurísticas, como é possível analisar na figura 16. Serão analisadas duas meta- heurísticas de solução única e quatro meta-heurísticas de população, acompanhadas por exemplos ilustrativos, para posteriormente determinar as que serão utilizadas no caso de estudo. Posteriormente será descrito o funcionamento das seguintes meta-heurísticas:

- *Simulated Annealing* (SA);
- *Tabu Search* (TS);
- *Ant Colony Optimization* (ACO);
- *Artificial Bee Colony* (ABC);
- *Genetic Algorithms* (GA);
- *Particle Swarm Optimization* (PSO).

3.6.1. *SIMULATED ANNEALING* (SA)

É uma meta-heurística baseada num método proposto, em 1953, por Metropolis et al. [32] e posteriormente definido por Kirkpatrick et al. [33] e Cerny [34]. É inspirada no processo de tratamento térmico dos materiais metálicos, que inicialmente são aquecidos até ao ponto de fusão e posteriormente arrefecidos lentamente para formar cristais metálicos, isto é, estruturas atómicas de baixa energia. O SA utiliza uma técnica semelhante em problemas de otimização de uma função de desempenho, utilizando uma variável *T*, representante da temperatura que serve para controlar os parâmetros da técnica.

O SA, partindo de uma solução inicial (habitualmente, aleatória) e inicializando *T* com um valor predeterminado, vai procurar, em cada iteração, por novas soluções na vizinhança da atual solução. Ao contrário da LS, no SA uma solução não necessita de ter um melhor desempenho para substituir a atual solução, isto é, mesmo soluções com pior desempenho podem ser aceites como próximas soluções, com uma probabilidade determinada pela expressão 20. Em cada iteração é escolhida uma solução aleatoriamente e posteriormente determinado estocasticamente se essa solução vai substituir a solução atual.

$$P(T, f(x_1), F(x)) = \exp\left(-\frac{f(x_1) - f(x)}{T}\right) \qquad (20)$$

A probabilidade de aceitar soluções com pior desempenho vai decrescer com *T*, tornando a meta-heurística mais seletiva durante o processo de pesquisa. Isto quer dizer que no início uma solução com pior desempenho terá muitas probabilidade de ser aceite como nova solução, fazendo uma pesquisa muito diversificada, mas durante o decursos da pesquisa essa probabilidade vai decrescendo, tornando a pesquisa mais intensa e seletiva no processo de seleção. No final da pesquisa a solução atual poderá não corresponder à melhor solução encontrada, assim é necessário manter não apenas a atual solução com a melhor solução encontrada até ao momento. O SA vai analisar um determinado número de soluções (L_n) para cada valor de *T*. Uma vez analisadas os L_n soluções *T* é diminuído, isto é, a temperatura baixa, tornando a pesquisa mais seletiva. Existem inúmeros métodos de arrefecimento. Considerando β um valor constante e α um valor constante onde $\alpha \epsilon$]0,1[, podem ser apresentando dois esquemas de arrefecimento [12,24,35]:

- Linear: $T = T - \beta$;
- Geométrico: $T = \alpha T$.

Existem também esquemas de arrefecimento adaptativos, isto é, dependentes do resultado da pesquisa. Já os critérios de interrupção são habitualmente relacionados com tempo ou temperatura, mas também podem interromper o SA depois de um número de iterações. O algoritmo do SA pode ser analisado na tabela 9, interrompido quando $T \leq T_F$.

Tabela 9 - *Simulated Annealing* [24]

Selecionar a solução inicial $x \in X$
Inicializar a temperatura $T = T_i$
while $T > T_F$
    repeat
        Selecionar aleatoriamente $x' \in v(x)$
        if $f(x') \leq f(x)$ **then**
            $x = x'$
        else
            $x = x'$ com probabilidade $p(T, f(x'), f(x))$
        end
    until tenham sido analisadas L_n soluções
    $T = T - \beta$
end
return melhor solução

Inicialmente é feita uma pesquisa quase aleatória, no entanto com o arrefecimento o SA torna-se mais seletivo. O desempenho da solução também vai influenciar a probabilidade de aceitação de uma solução, isto é, uma solução que piorar muito a atual solução têm poucas probabilidade de ser aceite, como pode ser analisado na figura 16.

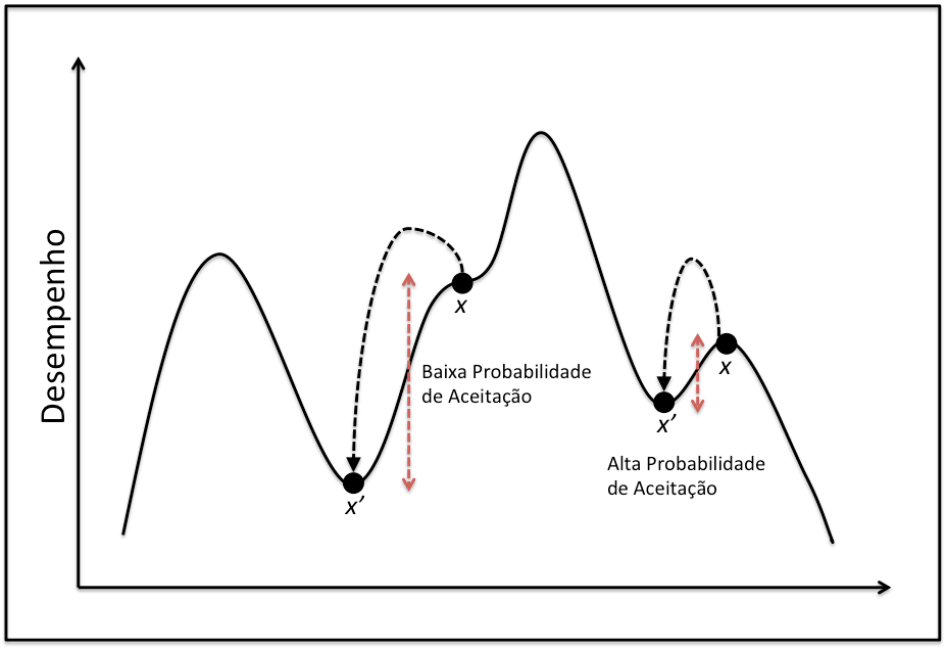

Figura 16 - Funcionamento do SA [adaptado de 12]

Para ilustrar o funcionamento do SA vai ser utilizado um problema de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, com seis tarefas, apresentado na tabela 7. Partindo da solução [1 2 3 4 5 6] com um valor de 48, com uma temperatura inicial de T_i = 20, um esquema de arrefecimento linear de $\beta = 5$, 5 iterações por temperatura ($L_n = 5$) e um critério de interrupção baseado na temperatura ($T \leq 10$). É importante notar que os parâmetros do SA foram escolhidos para um exemplo ilustrativo, que permita analisar o funcionamento desta meta-heurística. Na tabela 10 é analisado o funcionamento e desempenho do *Simulated Annealing* no problema em estudo.

Tabela 10 - Exemplo do SA [adaptado de 35]

	x'	*f(x')*	*x*	*f(x)*	*p(x')*	*p*
T = 20	1 2 4 3 5 6	39	1 2 3 4 5 6	48	-	-
	2 1 4 3 5 6	39	1 2 4 3 5 6	39	-	-
	2 1 4 5 3 6	42	2 1 4 3 5 6	39	0.8607	0.3708
	2 1 5 4 3 6	51	2 1 4 5 3 6	42	0.6376	0.7533
	2 4 1 5 3 6	42	2 1 4 5 3 6	42	-	-
T = 15	2 4 1 3 5 6	39	2 4 1 5 3 6	42	-	-
	2 4 3 1 5 6	38	2 4 1 3 5 6	39	-	-
	2 3 4 1 5 6	38	2 4 3 1 5 6	38	-	-
	2 3 1 4 5 6	46	2 3 4 1 5 6	38	0.5866	0.8727
	2 3 4 5 1 6	35	2 3 4 1 5 6	38	-	-
T = 10	2 3 5 4 1 6	44	2 3 4 5 1 6	35	0.4066	0.4762
	2 3 5 4 6 1	32	2 3 5 4 1 6	44	-	-
	2 3 5 6 4 1	44	2 3 5 4 6 1	32	0.3012	0.6676
	3 2 5 4 6 1	32	2 3 5 4 6 1	32	-	-
	3 2 5 6 4 1	44	3 2 5 4 6 1	32	0.3012	0.1754

Como é possível analisar, ao contrário do LS que resulta num ótimo local com valor de 36 (1 4 2 3 5 6) , o SA resulta em soluções com desempenho de 32 (2 3 5 4 6 1 e 3 2 5 4 6 1). É possível analisar que quando a temperatura baixa a probabilidade de aceitar soluções com piores desempenhos também baixa. Na 3ª iteração com T = 20 é aceite uma solução com pior desempenho, tal como acontece na 1ª e 5ª iteração com T = 10, do mesmo modo, na 4ª iteração com T = 20 , na 4ª iteração com T = 15 e na 3ª iteração com T = 10 as novas soluções com pior desempenho não são aceites como novas soluções [12,24,35].

3.6.2. *TABU SEARCH* (TS)

É uma meta-heurística proposta por Glover em 1986 [36], para problemas de otimização complexos. Ao contrário do SA, é uma meta-heurísticas dotada de memória, isto é, uma meta-heurísticas que utiliza as soluções passadas para orientar o processo de pesquisa futura. É inspirada no processo de memória humana através de uma lista de soluções ou movimentos proibidos, que vão permitir deslocar a pesquisa para além dos ótimos locais, pois não é possível voltar a uma solução que foi recentemente visitada. Isto vai introduzir diversidade, fazendo deslocar a pesquisa para espaços não explorados.

O TS, partindo de uma solução inicial (habitualmente aleatória) vai procurar em cada iteração por novas soluções na vizinhança da atual solução, escolhendo a que possui o melhor desempenho como próxima solução. O procedimento é semelhante ao da LS, mas ao contrário deste método, nem todas as soluções poderão ser consideradas como próximas soluções, apenas soluções que não pertençam à lista de soluções proibidas (lista tabu) deverão ser consideradas. O TS vai repetir esta pesquisa de vizinhança, atualizando a lista de soluções proibidas em cada interação, até o critério de interrupção ser encontrado.

Existem diversas formas de estruturar a memória do TS, podem ser armazenadas soluções proibidas ou apenas atributos das soluções, como uma determinada permutação de tarefas. Por exemplo uma solução 1 4 2 3 5 6 ou um determinado movimento $2 \rightarrow 4$ poderão ser colocados na lista tabu, proibindo soluções que resultem da troca das tarefas 2 e 4. Esta lista de movimentos proibidos evita que a pesquisa entre num ciclo infinito, tendo que aceitar soluções mesmo que tenham um desempenho pior que a atual solução. O tamanho da lista tabu vai influenciar o comportamento da meta-heurísticas. Listas pequenas resultam numa pesquisa intensa, muito focada numa pequena área do espaço de soluções, ao contrário listas maiores resultam em pesquisa muito diversificada, explorando uma maior área do espaço de pesquisa. O *Tabu Search* pode também utilizar listas tabu com tamanhos dinâmicos, utilizando inicialmente listas maiores para fazerem pesquisas muito diversificadas e depois reduzindo o tamanho para intensificar a pesquisa. O tamanho da lista também poderá ser adaptado dependendo dos resultados anteriores. Há movimentos que mesmo pertencentes à lista tabu devem ser considerados. Isto é feito através do critério de aspiração, que determina um patamar de desempenho para o qual movimentos mesmo que proibidos deverão ser considerados. Por exemplo não fazia sentido proibir uma troca entre $2 \rightarrow 4$ se resultasse na solução com o melhor desempenho, até ao momento.

Os critérios de interrupção estão habitualmente relacionados com o tempo e o número de iteração. O TS também pode ser interrompido ao fim de um determinado número de iterações sem melhoria do desempenho. O algoritmo do TS é apresentado na tabela 11.

Tabela 11 - *Tabu Search* [24]

```
Selecionar a solução inicial x ϵ X
Inicializar Lista Tabu com tamanho T
while ≠ Critério de Interrupção
        Selecionar melhor  x' ϵ v(x)
        if  x' ϵ Lista Tabu then
            if  f(x') = Critério de Aspiração then
                x ← x'
                Atualizar Lista Tabu
            else
                Remover x' de v(x)
            end
        else
            x ← x'
            Atualizar Lista Tabu
        end
end
return melhor solução
```

O TS utiliza essa lista de soluções ou movimentos proibidos para deslocar a pesquisa para além dos ótimos locais, como é possível analisar na figura 17. Senão fosse utilizada uma lista tabu para deslocar a pesquisa, ocorria num ciclo infinito entre as soluções *x* e *x'*.

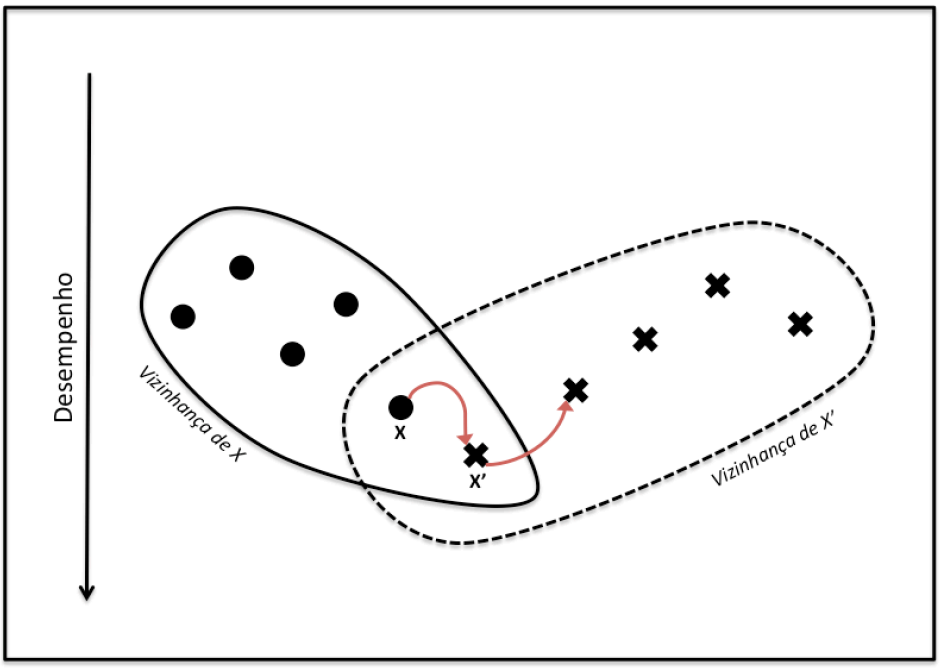

Figura 17 - Funcionamento do TS

Para ilustrar o funcionamento do TS vai ser utilizado um problema de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, com seis tarefas, apresentado na tabela 7. Partindo da solução [1 2 3 4 5 6] com um valor de 48, com uma lista tabu de dimensão 2 e um critério de interrupção baseado no número de iterações, isto é, interrompido ao fim de 6 iterações. É importante notar que os parâmetros do TS foram escolhidos para um exemplo ilustrativo, que permita analisar o funcionamento desta meta-heurística no exemplo, sem consideração pelo eficácia da meta-heurísticas. Na tabela 12 é possível analisar o funcionamento e desempenho do *Tabu Search* no problema exemplo.

Tabela 12 - Exemplo do TS [adaptado de 35]

	Solução	***f(x)***	***Lista Tabu***		***Solução***	***f(x)***	***Lista Tabu***
1ª Iteração	2 1 3 4 5 6	48	-	**2ª Iteração**	2 1 4 3 5 6	39	(3,4)
	1 3 2 4 5 6	50			**1 4 2 3 5 6**	**36**	
	1 2 4 3 5 6	**39**			-	-	
	1 2 3 5 4 6	51			1 2 4 5 3 6	42	
	1 2 3 4 6 5	51			1 2 4 3 6 5	42	
3ª Iteração	**4 1 2 3 5 6**	**37**	(3,4) (2,4)	**4ª Iteração**	-	-	(3,4) (2,4) (1,4)
	-	-			**4 2 1 3 5 6**	**39**	
	1 4 3 2 5 6	40			4 1 3 2 5 6	41	
	1 4 2 5 3 6	39			4 1 2 5 3 6	40	
	1 4 2 3 6 5	39			4 1 2 3 6 5	40	
5ª Iteração	-	-	(2,4) (1,4) (1,2)	**6ª Iteração**	2 4 3 1 5 6	38	(1,4) (1,2) (1,3)
	-	-			4 3 2 1 5 6	38	
	4 2 3 1 5 6	**38**			-	-	
	4 2 1 5 3 6	42			**4 2 3 5 1 6**	**35**	
	4 2 1 3 6 5	42			4 2 3 1 6 5	41	

Como é possível analisar, ao contrario do LS que fica bloqueado no ótimo local, o TS resulta numa solução com desempenho de 35 (4 2 3 5 1). Inicialmente o comportamento do TS é muito semelhante ao do LS, no entanto depois da lista tabu ser preenchida é possível analisar os movimentos que são proibidos, como por exemplo na 2ª iteração é proibida a permutação das tarefas 3 e 4. Isso vai permitir ultrapassar o ótimo local na 2ª iteração, evitando que ocorresse um ciclo infinito entre 1 4 2 3 5 6 e 4 1 2 3 5 6. O TS termina na 6ª iteração devolvendo a solução 4 2 3 5 1 6 com desempenho de 35 [12,24,35].

3.6.3. *ANT COLONY OPTIMIZATION* (ACO)

É uma meta-heurística proposta por Dorigo [37] inspirada no comportamento das formigas na procura de alimentos, que utilizam depósitos de feromonas para identificar os percursos mais promissores. O ACO inicialmente faz uma pesquisa diversificada, praticamente aleatória, mas rapidamente se intensifica, isto é, inicialmente cada indivíduo vai pesquisar aleatoriamente e depositando feromonas pelo percurso. Considerando que a feromona se evapora, os percursos mais curtos entre a colónia e a fonte de alimentos vão acumular uma maior concentração de feromona e assim atrair mais formigas.

Um problema de otimização combinatório pode ser representado por um grafo *G(V,E)*, onde os nós representam os componentes de uma solução e as arestas as ligações entres esses componentes [24]. Uma solução do problema corresponde ao percurso através de *G(V,E)*. No ACO cada formiga vai depositar feromona ao pesquisar através de *G(V,E)* de uma forma semi-aleatória, isto é, escolhendo o percurso através da feromona depositada e na expectativa do desempenho de cada nó. Os nós mais promissores vão acumular uma maior concentração de feromona resultando numa pesquisa mais intensa nessa área.

Embora existem muitas versões do ACO, todas apresentam procedimento semelhante. A meta-heurística habitualmente funciona através da iteração de três fases: construção de soluções, LS e atualização das feromonas. O ACO inicializa-se definindo os parâmetros, o número de indivíduos e a concentração de feromona inicial (τ_0). O número de indivíduos (m) controla as características da pesquisa; muitos indivíduos resultam numa pesquisa diversificada com poucas iterações, da mesma forma, muitos indivíduos resultam numa pesquisa intensa [38]. Posteriormente cada individuo partindo de uma solução vazia ($S_p=\varnothing$) e de um número finito elementos (C_{ij}), constrói uma solução adicionando passo à passo elementos de $N(S_p) \subseteq C$. A construção da solução deve considerar a concentração de feromona em cada ligações (τ_{ij}) e um método heurísticos para avaliar as expectativas do percurso, com uma componente estocástica. No *Ant System* (AS), uma variação do ACO, a probabilidade de escolher uma percurso é calculado pela expressão 21, onde p_{ij} representa a probabilidade do individuo acrescentar o elemento C_{ij} a solução [12,24,35].

$$p_{ij} = \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_k \tau_{ik}^{\alpha} * \eta_{ik}^{\beta}}, \qquad \forall\, C_{ij} \epsilon\, N(S_p) \tag{21}$$

O parâmetro τ_{ij} representa a concentração de feromona, enquanto η_{ij} representa a heurísticas do percurso C_{ij}. O η_{ij} deverá dar uma indicação do desempenho da inclusão de C_{ij}. No problema *Traveling Salesman Problem* (TSP), η_{ij} é inversamente proporcional à distância entre *i* e *j*, no problema de minimização dos atrasos ponderados em máquina única ($1||\Sigma w_j T_j$), η_{ij} é inversamente proporcional ao atraso resultante da introdução de C_{ij} na solução. Os parâmetros α e β controlam a importância relativa de τ_{ij} e η_{ij} na expressão 21. Posteriormente, poderá ser aplicada uma fase de LS, que é opcional e que resulta num aumento da intensidade do ACO. O ultimo passo é atualizar a concentração de feromona nos ramos de *G(V,E)*, através do depósito e evaporação de feromona. O depósito de feromona é normalmente realizado depois dos indivíduos terem construído soluções, tornando os ramos pertencentes às melhores soluções, mais atrativos. A evaporação de feromona deverá tornar os ramos anteriormente visitados menos atrativos, para introduzir diversidade na pesquisa. Existem muitas formas de atualizar a feromona, o que pode ser feito durante a construção das soluções ou depois de todos os indivíduos terem construído soluções. Uma das formas de atualizar a feromona pode ser analisado na expressão 22, onde ρ representa o coeficiente de evaporação que deverá estar compreendido entre $0 \leq \rho \leq 1$ e $1/f(S_p)$ o desempenho de uma solução do individuo *p*, apenas considerando no somatório os indivíduos que construíram soluções que incluam o ramo $C_{ij} \subseteq S_p$.

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + \sum_{p} \frac{1}{f(S_p)} \qquad (22)$$

Os critérios de interrupção são tipicamente relacionados com o tempo, número de iteração, ou quando todos os indivíduos percorrerem o mesmo percurso, isto é, quando todos os indivíduos tenham sido atraídos pela concentração de feromona de uma solução. É importante notar que existem inúmeras versões do ACO. Na tabela 13 é apresentado o algoritmo do ACO, que também é utilizado na versão AS.

Tabela 13 - *Ant Colony Optimization* [24]

Inicializar valores de feromona
while ≠ *Critério de Interrupção*
Construção de Soluções
Pesquisa Local
Atualização da Feromona
end
return melhor solução

Para ilustrar o funcionamento do ACO, particularmente do AS, foi utilizado um problema de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, com seis tarefas, apresentado na tabela 7. Partindo de soluções aleatórias, uma população de 4 indivíduos, feromona arredondada às décimas e inicializada (τ_0) a 0.01, um coeficiente de evaporação (ρ) de 0.5, α=1 e β=1 e interrompido depois de 3 iterações. É importante notar que os parâmetros do AS foram escolhidos para analisar o funcionamento desta meta-heurística sabendo que eram necessários mais indivíduos e iterações para encontrar soluções eficientes. Na tabela 14 é possível analisar o funcionamento do *Ant System* no exemplo.

Tabela 14 - Exemplo do AS [adaptado de 35]

	m	***Solução***	***f(x)***	***Concentração de Feromonas***
1ª Iteração	1	3 2 5 4 1 6	44	
	2	6 5 2 4 1 3	57	
	3	4 2 3 5 1 6	35	
	4	4 6 2 1 5 3	50	
2ª Iteração	1	4 2 3 5 6 1	23	
	2	6 5 2 3 4 1	47	
	3	4 2 5 3 6 1	23	
	4	3 1 6 5 4 2	73	
3ª Iteração	1	4 2 3 5 6 1	23	
	2	3 5 2 4 1 6	44	
	3	4 1 6 5 2 3	55	
	4	4 2 5 3 6 1	23	

	1	**2**	**3**	**4**	**5**	**6**
1	0.01	0.01	0.03	0.01	0.03	0.06
2	0.03	0.01	0.04	0.03	0.03	0.01
3	0.01	0.03	0.03	0.04	0.01	0.01
4	0.05	0.04	0.01	0.06	0.01	0.03
5	0.04	0.03	0.03	0.03	0.01	0.01
6	0.01	0.03	0.01	0.01	0.03	0.03

	1	**2**	**3**	**4**	**5**	**6**
1	0.01	0.01	0.02	0.01	0.02	0.04
2	0.02	0.01	0.08	0.02	0.06	0.01
3	0.02	0.02	0.03	0.04	0.05	0.05
4	0.05	0.12	0.01	0.12	0.01	0.02
5	0.02	0.04	0.06	0.03	0.01	0.05
6	0.09	0.02	0.01	0.01	0.05	0.04

	1	**2**	**3**	**4**	**5**	**6**
1	0.01	0.01	0.01	0.01	0.01	0.06
2	0.01	0.01	0.10	0.03	0.07	0.01
3	0.01	0.01	0.04	0.02	0.09	0.07
4	0.07	0,15	0.01	0.17	0.01	0.01
5	0.01	0.06	0.07	0.02	0.01	0.07
6	0.13	0.01	0.01	0.01	0.04	0.02

Foram encontradas duas soluções com valor de 23 [4 2 3 5 6 1] e [4 2 5 3 6 1]. A evolução da concentração de feromonas resulta numa maior probabilidade de incluir certos ramos. Por exemplo, na 2ª iteração há 40% de probabilidade de escolher a 4ª tarefa em primeiro, 55% na 3ª iteração e 65% numa suposta 4ª iteração. Esta probabilidade vai aumentar até todos os indivíduos percorrerem esse ramo e diminuir quando for encontrada uma solução com melhor resultado que não inclua a 4ª tarefa em primeiro [12,24,35].

3.6.4. *ARTIFICIAL BEE COLONY* (ABC)

Existem diversas meta-heurísticas inspiradas pelo comportamento de abelhas, a *Queen-Bee Evolution* (QBE), baseada na estrutura da colónia, a *Marriage in Honey Bees Optimization* (MBO) no processo de reprodução, e *Bee Colony Optimization* (BCO) ou *Virtual Bee Algorithm* (VBA), na procura de comida [24]. O *Artificial Bee Colony* (ABC) é uma meta-heurística, proposta por Karaboga [39] e Pham et al. [40] em 2005. É uma técnica de optimização inspirada no comportamento de uma colónia de abelhas na procura por alimentos, dividindo as abelhas em abelhas trabalhadoras, abelhas oportunistas e abelhas exploradoras, com características diferentes. É importante notar que esta meta-heurísticas era apenas aplicável em problemas de otimização contínuos, não podendo ser aplicada em problemas de otimização discretos, como por exemplo, os problemas de escalonamento. Recentemente foram propostas versões do ABC para problemas discretos, *Discrete Artificial Bee Colony* (DABC), incluindo uma versão proposta por Karaboga & Gorkemli em 2011, para abordar o TSP [41]. Outras versões do DABC podem ser analisadas em [42-44] com características diferentes da proposta em [41]. O DABC tem um procedimento praticamente idêntico ao ABC, apenas alterando a forma como as abelhas se movimentam na vizinhança de uma fonte de alimentos, que é realizada através da manipulação das soluções, transformando-as em soluções vizinhas.

A colónia é composta por um dado número de abelhas, em que habitualmente, metade são trabalhadoras/exploradoras e metade abelhas oportunistas. As abelhas trabalhadoras vão explorar as fontes de alimento (soluções) e comunicar os resultados ao resto das abelhas. Posteriormente as abelhas oportunistas vão escolher uma fonte de alimentos, de uma forma probabilística com base na quantidade de alimento (desempenho) de cada fonte. Quando as fontes de alimentos forem exaustas, as abelhas trabalhadoras vão transformar-se em abelhas exploradoras e procurar por novas fontes de alimento.

O DABC funciona através da iteração de três fases, enquanto não ocorrer o critério de interrupção: a fase das abelhas trabalhadoras, a fase das abelhas oportunistas e a fase das abelhas exploradoras. Inicialmente é determinada, habitualmente de forma aleatória ou de forma parcialmente aleatória com uma componente heurística, uma fonte de alimento (x_i) para cada abelha trabalhadora. Na fase das abelhas trabalhadoras, cada uma vai explorar uma solução na vizinhança (v_i) da sua fonte de alimento, se obtiver um desempenho superior à da atual fonte de alimento, então deverá substituir essa fontes de alimento.

Posteriormente ocorre a fase das abelhas oportunistas, que esperam pelo desempenho de cada fonte de alimento para determinar que fonte de alimento vão explorar. Isto quer dizer que as abelhas oportunistas esperam na colónia pela informação disponibilizada pelas abelhas trabalhadoras para escolher, probabilisticamente, as fontes de alimento. Na expressão 23, é apresentada uma forma de calcular as probabilidades, onde f_i representa o desempenho da fonte de alimento x_i [44]. Em problemas de minimização o desempenho deverá ser representado por $1/f_i$, por exemplo no problema em estudo, poderá ser utilizado $1/w_jT_j$. Após escolher uma fonte de alimentos cada abelha oportunista vai explorar uma solução na vizinhança e se obtiver um desempenho superior ao da atual fonte de alimentos, então deverá substituir essa fonte de alimento.

$$p_i = \frac{f_i}{\sum f_i} \tag{23}$$

Finalmente, ocorre a fase das abelhas exploradoras, depois de uma fonte de alimentos ser abandonada. Uma fonte de alimento é abandonada depois de ocorrerem l iterações sem melhoria do desempenho. Karaboga & Gorkemli em [41] apresentam valores indicativos do número de iterações sem melhoria que devem ocorrer antes de uma fonte de alimento ser abandonada, sabendo que l maiores resultam em pesquisas mais intensas enquanto l mais pequenos resultam em pesquisas com maior diversidade. Se uma fonte de alimento for abandonada, a abelha trabalhadora dessa fonte de alimento vai transforma-se numa abelha exploradora e mover-se para uma nova fonte de alimento, habitualmente, de forma aleatória ou de forma parcialmente aleatória. Na tabela 15 é apresentada a estrutura habitual do ABC que é idêntica ao do DABC [12].

Tabela 15 - *Artificial Bee Colony* [24]

```
Inicializar fontes de alimentos
while ≠ Critério de Interrupção
    Fase das Abelhas Trabalhadoras
    Fase das Abelhas Oportunistas
    if fonte de alimentos exausta
        Fase das Abelhas Exploradoras
    end
end
return melhor solução
```

Os critérios de interrupção são habitualmente relacionados com o tempo, o número de iterações, ou depois de explorado um determinado número de fontes de alimentos.

Para ilustrar o funcionamento do DABC, foi utilizado um problema de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, com seis tarefas, apresentado na tabela 7. Partindo de fontes de alimento iniciais aleatórias, uma população de 4 abelhas, 2 abelhas trabalhadoras/exploradoras e 2 abelhas oportunistas. Considerando que uma fonte de alimentos é totalmente explorada ao fim de duas iterações sem melhoria ($l=2$) e interrompida depois de 6 iterações. O mecanismo de vizinhança (v_i) é baseado na troca de pares adjacentes, como aconteceu nos exemplos das meta-heurísticas anteriores. Na tabela 16 é possível analisar o funcionamento do DABC no exemplo ilustrativo.

Tabela 16 - Exemplo do DABC

	Fontes e Abelhas	**Soluções**	***f(x)***		**Fontes e Abelhas**	**Soluções**	***f(x)***
1ª Iteração	F. de Alimentação A	3 4 5 1 2 6	49	**2ª Iteração**	F. de Alimentação A	3 4 5 2 1 6	39
	A. Trabalhadora	3 5 4 1 2 6	49		A. Trabalhadora	4 3 5 2 1 6	39
	A. Oportunista	3 4 5 2 1 6	39		A. Oportunista	3 5 4 2 1 6	42
	F. de Alimentação B	2 3 6 4 1 5	50		A. Oportunista	3 4 2 5 1 6	35
	A. Trabalhadora	2 3 6 4 5 1	41		F. de Alimentação B	2 3 6 4 5 1	41
	A. Oportunista	2 3 6 1 4 5	64		A. Trabalhadora	2 6 3 4 5 1	44
3ª Iteração	F. de Alimentação A	3 4 2 5 1 6	35	**4ª Iteração**	F. de Alimentação A	3 4 2 5 1 6	35
	A. Trabalhadora	3 2 4 5 1 6	35		A. Trabalhadora	3 4 5 2 1 6	39
	A. Oportunista	4 3 2 5 1 6	35		A. Oportunista	3 4 2 5 6 1	23
	A. Oportunista	3 4 2 1 5 6	38		A. Oportunista	3 2 4 5 1 6	35
	F. de Alimentação B	2 3 6 4 5 1	41		F. de Alimentação C	5 2 4 6 1 3	45
	A. Trabalhadora	2 6 3 4 5 1	41		A. Trabalhadora	5 2 4 6 3 1	32
5ª Iteração	F. de Alimentação A	3 4 2 5 6 1	23	**6ª Iteração**	F. de Alimentação A	3 4 2 5 6 1	23
	A. Trabalhadora	3 4 5 2 6 1	27		A. Trabalhadora	3 4 2 5 1 6	35
	A. Oportunista	3 2 4 5 6 1	23		A. Oportunista	4 3 2 5 6 1	23
	F. de Alimentação C	5 2 4 6 3 1	32		A. Oportunista	3 4 5 2 6 1	27
	A. Trabalhadora	5 4 2 6 3 1	32		F. de Alimentação C	5 2 4 6 3 1	32
	A. Oportunista	5 2 6 4 3 1	44		A. Trabalhadora	5 2 4 3 6 1	23

Foram encontradas duas soluções de valor 23. Existe uma maior probabilidade das abelhas oportunistas escolherem as fontes de alimentos com melhor desempenho, por exemplo na 2ª iteração, a fonte de alimentos A é escolhida em 51% dos casos. É de notar que a fonte de alimento B é exausta na 3ª iteração e a fonte A na 6ª iteração [24,41-44].

3.6.5. *GENETIC ALGORITHMS*

É uma das técnicas evolutivas mais utilizadas, proposta por Holland [45] em 1972, baseada na Teoria da Evolução de Charles Darwin. Os GA funcionam através do cruzamento de indivíduos da população habitualmente de uma forma elitista, para encontrar indivíduos mais aptos, isto é, soluções com melhor desempenho. Partindo de uma população, os indivíduos são avaliados (*fitness*) e são escolhidos os que possuem melhores características para cruzamento, na expectativa que os descendentes herdem as características dos seus ascendentes e resultem em soluções com melhor desempenho. Já os indivíduos menos aptos têm menor probabilidade de serem cruzados, na expectativa de eliminar da população as características que resultem em soluções com pior desempenho.

Representando as soluções por cromossomas, que retratam as características dos indivíduos, os GA selecionam indivíduos para cruzamento (*crossover*) e fazem mutações (*mutation*) para manter diversidade na pesquisa. Partindo de uma população de *N* indivíduos são selecionados os indivíduos com melhor desempenho para cruzamento, isto é, os indivíduos com melhor desempenho são escolhidos automaticamente (*Greedy*) ou é utilizado um mecanismo estocástico com uma componente elitista, fazendo com que os indivíduos com melhores características tenham mais probabilidade de serem escolhidos. Os descendentes devem substituir, totalmente ou parcialmente, os indivíduos da atual população, de maneira a que o tamanho da população se mantenha constante.

Dependendo da representação escolhida existem muitos mecanismos de cruzamento para representações de permutações utilizadas em problema de escalonamento, os mecanismos de cruzamento mais utilizados são o *Order Crossover* (OX), *Partially Mapped Crossover* (PMX) e *Two-Point Crossover* que podem ser analisados na figura 18 [12,35].

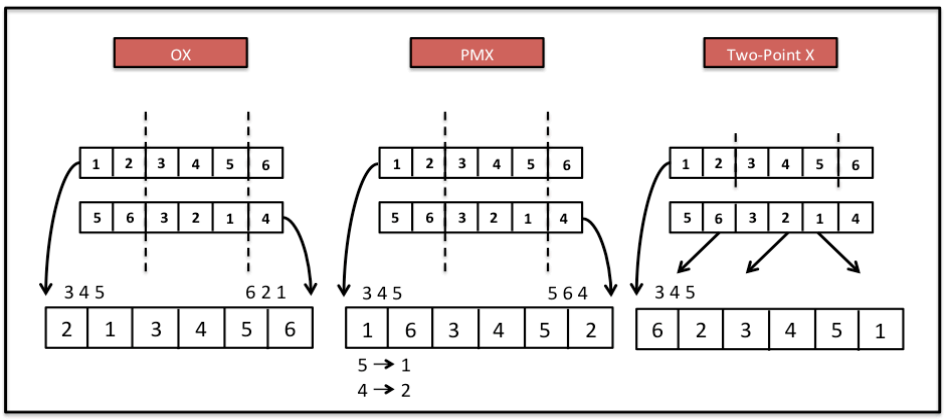

Figura 18 -Mecanismos de *Crossover*

Os indivíduos escolhidos não serão todos cruzados, p_c representa a taxa de *crossover* que é a probabilidade de um par de indivíduos serem cruzados numa iteração. Os valores mais habituais para p_c situam-se entre [0.45,0.95], embora também possam ser utilizados taxas de *crossover* adaptativas, que controlam as características durante o processo da pesquisa.

Os mecanismo de mutação (*mutation*) permitem introduzir maior diversidade na população e consequentemente na pesquisa, fazendo alterações num pequeno número de indivíduos. Sendo p_m a taxa de mutação, com valores habitualmente entre [0.001,0.01], ou com uma probabilidade de *1/k* onde *k* representa o número de variáveis de decisão, resultando, em média, apenas numa variável alterada por iteração. Os mecanismos de mutação deverão permitir aos GAs explorar a totalidade do espaço de soluções, movimentando a pesquisa para além da atual vizinhança. O tipo de mutação depende da representação escolhida, em representações de permutações, habituais em problema de escalonamento, os mecanismos de mutação mais utilizados são, *swapping*, *inversion* ou *insertion*. O *insertion* corresponde à escolha de uma tarefa e o movimento para uma posição aleatória. Por exemplo, na sequência [1 2 3 4 5 6] a tarefa 2 é movida para a 5, resultando em [1 3 4 5 2 6] [12].

Os critérios de interrupção são tipicamente relacionados com o tempo de processamento ou número de iterações (*generations*) estático ou adaptativo, quando a pesquisa é interrompida depois de um número de iterações sem melhoria da solução. É importante notar que existem muitas variações do GAs, na tabela 17 é apresentando o algoritmo clássico do GA.

Tabela 17 - *Genetic Algorithms* [35]

```
Inicializar população inicial
Generation = 0
while ≠ Critério de Interrupção
    Generation = Generation + 1
    Fitness da população
    Seleção dos indivíduos
    if < pc
        Crossover
    end
    if < pm
        Mutation
    end
    Substituição da população
end
return melhor solução
```

Para ilustrar o funcionamento do GAs é utilizado um problema de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, com seis tarefa, apresentado na tabela 6. Partindo de soluções aleatórias, uma população de 4 indivíduos substituída totalmente em cada iteração, com seleção dos 3 melhores indivíduos para cruzamento por *Two-Point Crossover*, com taxa de cruzamento de 100% (p_c=1), uma taxa de mutação de 5% (p_m=0.05) por *insertion* e interrompido ao fim de 6 iterações. É importante notar que os parâmetros do GA foram selecionados para analisar o funcionamento desta meta-heurística. Note-se que, foi escolhido uma taxa de mutação alta para se poder analisar no exemplo. Na tabela 18, é possível analisar o *Genetic Algorithm* no problema em estudo.

Tabela 18 - Exemplo do GA [adaptado de 35]

	Pais	**População**	***f(x)***		**Pais**	**População**	***f(x)***
1ª Iteração	-	**4 3 2 5 1 6**	35	**2ª Iteração**	3 2 5 4 6 1	**3 2 5 4 6 1**	32
	-	**1 4 2 3 6 5**	39		4 3 2 5 1 6	**3 4 2 5 1 6**	35
	-	2 3 6 4 5 1	41		3 2 5 4 6 1	3 2 5 1 4 6	58
	-	**3 2 5 4 6 1**	32		1 4 2 3 6 5	**1 4 2 3 5 6**	36
3ª Iteração	3 2 5 4 6 1	**3 2 5 4 1 6**	44	**4ª Iteração**	3 4 2 5 6 1	**4 2 3 5 6 1**	23
	3 4 2 5 1 6	**3 4 2 5 6 1**	23		4 6 2 3 5 1	**4 2 6 3 5 1**	29
	3 2 5 4 6 1	1 2 5 4 6 3	54		3 4 2 5 6 1	3 4 2 5 1 6	35
	1 4 2 3 5 6	**4 6 2 3 5 1**	29		3 2 5 4 1 6	**3 2 5 4 6 1**	32
5ª Iteração	4 2 3 5 6 1	2 3 5 6 4 1	44	**6ª Iteração**	4 2 3 5 6 1	4 2 3 5 6 1	23
	4 2 6 3 5 1	**4 2 6 3 5 1**	29		4 2 6 3 5 1	4 2 6 3 5 1	29
	4 2 3 5 6 1	**4 2 3 5 6 1**	23		4 2 3 5 6 1	4 2 3 5 6 1	23
	3 2 5 4 6 1	**3 2 5 4 6 1**	32		3 2 5 4 6 1	3 2 5 4 6 1	32

Foram encontradas duas soluções com valor de 23 [3 4 2 5 6 1] e [4 2 3 5 6 1]. Uma análise do desempenho do GA demonstra a evolução do processo de pesquisa para essas soluções e uma população muito semelhante. Por exemplo, depois da 3ª iteração é possível notar que cruzamentos entre as soluções mais eficientes resulta sempre em soluções muito semelhantes, não permitindo afastar a pesquisa para outras áreas do espaço de soluções. Isto sucede mesmo depois de acontecer uma mutação na 4ª iteração, onde o primeiro indivíduo sofre uma mutação, movendo a tarefa 4 da 1ª para 5ª posição [12,24,35].

3.6.6. *PARTICLE SWARM OPTIMIZATION*

O PSO é uma meta-heurística proposta por Kennedy & Eberhart [46] em 1995, inspirada em bandos de pássaros ou cardumes de peixes. É baseada no conhecimento e capacidade de decisão de cada partícula e no conhecimento da nuvem de partículas. No PSO cada partícula representa uma solução, que se vai movimentar através do espaço de pesquisa *N*-dimensional, onde *N* corresponde ao tamanho do problema, com velocidade variável. O movimento de cada partícula vai basear-se no conhecimento da partícula e no conhecimento da nuvem de partículas. Assim, cada partícula mantém em memória a solução com melhor desempenho que encontrou até ao momento, *lbest* (*Local Best*), e a melhor solução encontrada pela nuvem de partículas *gbest* (*Global Best*).

É inicializado através da seleção, habitualmente aleatória, da posição e velocidade das partículas e determinada a partícula com melhor desempenho (*gbest*). Posteriormente são atualizadas as velocidades de todas as partículas que serão atraídas por *lbest* e *gbest*. Cada partícula têm uma velocidade diferente, calculada através da expressão 24, onde V_{id} representa a velocidade da partícula *i* na dimensão *d*, X_{id} a posição da partícula, C_1 e C_2 os coeficientes de aceleração em *lbest e gbest* respectivamente e φ_1 e φ_2 valores aleatórios compreendidos entre [0;1]. Os valores de C_1 e C_2 permitem controlar as características do PSO, valores de C_1 mais elevados resultam numa maior atração de *lbest*, enquanto valores de C_2 mais elevados resultam numa maior atração de *gbest*.

$$V_{id}(t+1) = V_{id}(t) + \varphi_1 C_1\big(lbest - X_{id}(t)\big) + \varphi_2 C_2\big(gbest - X_{id}(t)\big) \qquad (24)$$

É necessário calcular as velocidades das partículas em todas as dimensões, por exemplo no problema de dimensão 5, com 5 partículas é necessário calcular 25 velocidades. Poderá ser considerado um limite da velocidade das partículas, para evitar que as partículas se desloquem muito rapidamente através do espaço de pesquisa. Os limites mais apertados na velocidade, [V_{min};V_{max}], resultam numa pesquisa mais intensa [24,35]. Após calculadas as velocidades é necessário atualizar as posições das partículas, que se deslocam da posição $X_{id}(t)$ para $X_{id}(t+1)$, através da expressão 25. As partículas vão aproximando-se de *lbest* e *gbest*, isto é, as partículas são atraídas pelas melhores soluções que conhecem e aquelas que são do conhecimento colectivo da nuvem de partículas.

$$X_{id}(t+1) = X_{id}(t) + V_{id}(t+1) \qquad (25)$$

Uma atualização da expressão da velocidade (24), pode ser analisada na expressão 26, onde foi acrescentado um componente de inércia (ω) que permite controlar os parâmetros da pesquisa, introduzindo um componente da velocidade anterior. Os valores da inércia são habitualmente uma constante positiva ou uma função que decresce no tempo [24,35]. Os valores de inércia muito elevados resultam em pesquisas muito diversificada, enquanto valores de inércia baixos resultam em pesquisas com mais intensidade, podendo paralisar o PSO num ótimo local. Habitualmente devem ser utilizados valores da inércia que permitam fazer pesquisas eficientes do espaço de pesquisa. Outra forma de controlar o impacto da velocidade através de um coeficiente de constrição é apresentando em [47].

$$V_{id}(t+1) = \omega\, V_{id}(t) + \varphi_1 C_1\big(lbest - X_{id}(t)\big) + \varphi_2 C_2\big(gbest - X_{id}(t)\big) \qquad (26)$$

Os critérios de interrupção do PSO estão tradicionalmente relacionados com o tempo de processamento, número de iterações pré-definido ou um número de iterações adaptativo, interrompendo o PSO depois de um número pré-estabelecido de iterações sem melhoria da melhor solução (*gbest*). Na tabela 19 é apresentado o algoritmo do PSO.

Tabela 19 - *Particle Swarm Otptimization* [24]

```
Inicializar X_id(0) e V_id(0) da população de partículas
while ≠ Critério de Interrupção
    for cada partícula
        Atualizar V_id(t)
        Atualizar X_id(t)
        If f(X_id) melhor que f(lbest)
            lbest = X_id
        end
        If f(X_id) melhor que f(gbest)
            gbest = X_id
        end
    end
end
return melhor solução
```

O procedimento do PSO é apropriado para problemas de otimização contínua, no entanto é possível utilizar esquemas de representação para problemas de otimização discretas. Em [35] é apresentada uma forma de representação para problemas de escalonamento. Nessa representação as posições das partículas são transformadas numa permutação, onde a tarefa 1 é representada pelo X_{id} mais pequeno de todas as dimensões. Por exemplo uma partícula em $X_{id} = \{0.23;0.34;0.03;0.12;0.08\}$ corresponde à solução [4 5 1 3 2].

Para ilustrar o funcionamento do PSO é utilizando um problema de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, com seis tarefas, apresentado na tabela 6. Partindo de soluções aleatórias, uma população de 4 partículas, com posições iniciais ($X_{id}(0)$) aleatórias, sem velocidade inicial ($V_{id}(0) = 0$), com valores de C_1 e C_2 semelhantes ($C_1 = C_2 = 1$), coeficiente de inércia constante ($\omega = 0.5$) interrompido de 4 iterações. No exemplo os valores de X_{id} e V_{id} foram arredondados às centésimas. É importante notar que os parâmetros do PSO foram escolhidos para analisar o funcionamento desta meta-heurística num pequeno problema; eram necessárias mais partículas, iterações e outros parâmetros para fazer uma pesquisa mais diversificada e encontrar soluções eficientes. Na tabela 20 é possível analisar o *Particle Swarm Optimization* no exemplo.

Tabela 20 Exemplo do PSO [adaptado de 35]

	V_{id}	X_{id}	Soluções	*f(x)*	*lbest*
1ª Iteração	0 0 0 0 0 0	0.23 0.13 0.29 0.33 0.42 0.49	2 1 3 4 5 6	48	**0.23 0.13 0.29 0.33 0.42 0.49**
	0 0 0 0 0 0	0.39 0.12 0.14 0.21 0.26 0.23	6 1 2 3 5 4	74	0.39 0.12 0.14 0.21 0.26 0.23
	0 0 0 0 0 0	0.38 0.06 0.24 0.45 0.05 0.28	5 2 3 6 1 4	58	0.38 0.06 0.24 0.45 0.05 0.28
	0 0 0 0 0 0	0.35 0.41 0.13 0.48 0.22 0.18	4 5 1 6 3 2	58	0.35 0.41 0.13 0.48 0.22 0.18
2ª Iteração	0 0 0 0 0 0	0.23 0.13 0.29 0.33 0.42 0.49	2 1 3 4 5 6	48	0.23 0.13 0.29 0.33 0.42 0.49
	-0.05 0 0.08 0.04 0.02 0.19	0.34 0.12 0.22 0.25 0.28 0.42	5 1 2 3 4 6	54	0.34 0.12 0.22 0.25 0.28 0.42
	-0.15 0.04 0.02 -0.04 0.17 0.18	0.23 0.10 0.26 0.40 0.22 0.46	3 1 4 5 2 6	37	**0.23 0.10 0.26 0.40 0.22 0.46**
	-0.11 -0.17 0.04 -0.05 0.04 0.06	0.24 0.24 0.17 0.43 0.26 0.24	2 3 1 6 5 4	62	0.35 0.41 0.13 0.48 0.22 0.18
3ª Iteração	0 -0.02 -0.02 0.03 -0.09 -0.01	0.23 0.11 0.27 0.36 0.33 0.48	2 1 3 5 4 6	51	0.23 0.13 0.29 0.33 0.42 0.49
	-0.11 -0.02 0.07 0.11 0 0.11	0.23 0.10 0.29 0.36 0.28 0.53	2 1 4 5 3 6	42	0.23 0.10 0.29 0.36 0.28 0.53
	-0.08 0.02 0.01 -0.02 0.09 0.09	0.15 0.12 0.27 0.38 0.31 0.55	2 1 3 5 4 6	51	**0.23 0.10 0.26 0.40 0.22 0.46**
	0.02 -0.11 0.08 -0.01 -0.03 0.08	0.26 0.12 0.25 0.42 0.23 0.32	4 1 3 6 2 5	50	0.26 0.12 0.25 0.42 0.23 0.32
4ª Iteração	0 0 -0.01 0.01 -0.06 -0.01	0.23 0.11 0.26 0.37 0.27 0.47	2 1 3 5 4 6	51	0.23 0.13 0.29 0.33 0.42 0.49
	-0.06 -0.01 0.03 0.07 -0.03 0	0.17 0.09 0.32 0.43 0.25 0.53	2 1 4 5 3 6	42	0.23 0.10 0.29 0.36 0.28 0.53
	0.07 -0.02 -0.01 0.01 -0.06 -0.01	0.22 0.10 0.26 0.39 0.25 0.54	2 1 4 5 3 6	42	**0.23 0.10 0.26 0.40 0.22 0.46**
	-0.02 -0.07 0.05 -0.02 -0.02 0.06	0.24 0.05 0.30 0.40 0.21 0.38	3 1 4 6 2 5	63	0.26 0.12 0.25 0.42 0.23 0.32

Foi encontradas uma solução com desempenho de 37 (3 1 4 5 2 6). No problema exemplo é possível analisar a influência da 3ª partícula nas restantes, depois da 2ª iteração, resultando em soluções com características semelhantes. No exemplo é possível analisar como as velocidades das partículas diminuem ao aproximarem-se das melhores soluções [12,24,35].

3.7. PARAMETRIZAÇÃO DE META-HEURÍSTICAS

Os parâmetros de uma meta-heurística têm muita influência na qualidade das soluções encontradas. Em [47] é apresentada uma definição formal do processo de parametrização: "*Para uma técnica de otimização A, uma instância do problema I, com um índice de desempenho C, quais são os parâmetros de A que otimizam C em I*". O desempenho de uma meta-heurística é habitualmente uma função da qualidade das soluções encontradas e o tempo computacional necessário. O processo de parametrização é habitualmente moroso, sendo uma parte importante do esforço de aplicação de uma meta-heurística [48]. Em Montero et al [49] os processos de parametrização de meta-heurísticas são divididos em:

- **Parametrização Manual** - Os parâmetros são alterados iterativamente pelo utilizador, na procura por uma combinação de parâmetros que resulte no melhor desempenho da técnica de otimização, para determinado problema.
- **Parametrização por Analogia** - Os parâmetros são definidos pela expectativa do desempenho numa instância do problema, através da identificação do impacto dos parâmetros nas características da técnica de otimização.
- **Parametrização por Planeamento de Experiências** - Os parâmetros são definidos estatisticamente, através de uma análise dos valores de parâmetros no desempenho da técnica de otimização, num determinado problema.
- **Parametrização por Pesquisa** - É feita uma pesquisa pelo espaço de parâmetros, através de LS ou outras meta-heurísticas, na procura dos parâmetros que resultam no melhor desempenho num determinado problema.
- **Parametrização Híbrida** - Os parâmetros são definidos através de uma mistura de técnicas de parametrização. Por exemplo, pela combinação de parametrização por Planeamento de Experiências e Parametrização por Pesquisa [48].

Em [49] também são apresentadas várias técnicas de parametrização: *F-Race* [50], *Relevance Estimation and Value Calibration* (REVAC) [51], *Parameter Iterated Local Search* (ParamILS) [52] e o *Sequential Parameter Optimization* (SPO) [53]. O autor conclui com indicações na seleção das técnicas de parametrização, por exemplo demonstrando que o *F-Race* é o método que necessita de mais tempo computacional e apresenta o ParamILS e o SPO como as técnicas mais simples. Outras técnicas de parametrização podem ser analisadas em [48], onde é apresentada uma técnica híbrida, ou em [47], onde é apresentada uma versão mais recente do SPO.

3.7.1. PLANEAMENTO DE EXPERIÊNCIAS DE TAGUCHI

Uma das técnicas de parametrização mais utilizadas, parametrização por planeamento de experiências (*Design of Experiments* - DOE), demonstra ser muito ineficiente, particularmente quando é realizada uma experiencia factorial completa, tornando-se muito morosa quando existem muitos parâmetros e níveis. O planeamento de experiências de Taguchi utiliza matrizes balanceadas para analisar um número elevado de parâmetros e níveis em experiências [54]. Enquanto uma experiência factorial completa de 4 parâmetros e 3 níveis necessitava de 81 (3^4) experiências, o planeamento de experiências de Taguchi apenas necessita de 9 (1+8) experiências, através da matriz balanceada L_9. Isto torna-se mais evidente quando há uma repetição das experiências, tornando os números de experiências mais desnivelados, por exemplo, numa experiência factorial completa de 4 parâmetros, 3 níveis e 3 repetições são necessárias 243 experiências, enquanto que no planeamento de experiências de Taguchi são apenas necessárias 27 experiências [54].

Enquanto o planeamento de experiências factorial completo utiliza apenas medidas para analisar o impacto dos parâmetros, o planeamento de experiências de Taguchi combina as médias com as variâncias numa única medida de desempenho S/N. O S/N deverá ser calculado pela expressão 27, para problemas de minimização. O desempenho do parâmetro num determinado nível é representado por y_i e n o número de observações.

$$\mathrm{S/N} = -10\log\left(\frac{1}{n}\sum_{i=1}^{n} y_i^2\right) \qquad (27)$$

Para ilustrar o funcionamento do planeamento de experiências de Taguchi foi utilizada uma meta-heurística com 4 parâmetros (P_1, P_2, P_3, P_4) e 3 níveis para cada parâmetro, representados na tabela 21. Neste caso, serão necessárias 9 experiências (1+8), sendo utilizada a matriz balanceada L_9. No exemplo não serão utilizados valores reais para se compreender o procedimento do planeamento de experiências de Taguchi.

Tabela 21 - Parâmetros do Exemplo

	1	**2**	**3**
P₁	P_{11}	P_{12}	P_{13}
P₂	P_{21}	P_{22}	P_{23}
P₃	P_{31}	P_{32}	P_{33}
P₄	P_{41}	P_{42}	P_{43}

Posteriormente, é utilizada a matriz L_9 para determinar que experiências vão ser realizadas, como pode ser analisado na tabela 22. Os níveis de cada parâmetro devem ser substituídos na matriz balanceada que permite experimentar cada parâmetro o mesmo número de vezes. Os resultados das 9 experiências são representados por Z_i.

Tabela 22 - S/N das Experiências do Exemplo

	P_1	P_2	P_3	P_4	$f(x)$	S/N
1	$P_{11}(1)$	$P_{21}(1)$	$P_{31}(1)$	$P_{41}(1)$	Z_1	$-10 \log (Z_1^2)$
2	$P_{11}(1)$	$P_{22}(2)$	$P_{33}(3)$	$P_{42}(2)$	Z_2	$-10 \log (Z_2^2)$
3	$P_{11}(1)$	$P_{23}(3)$	$P_{32}(2)$	$P_{43}(3)$	Z_3	$-10 \log (Z_3^2)$
4	$P_{12}(2)$	$P_{21}(1)$	$P_{33}(3)$	$P_{43}(3)$	Z_4	$-10 \log (Z_4^2)$
5	$P_{12}(2)$	$P_{22}(2)$	$P_{32}(2)$	$P_{41}(1)$	Z_5	$-10 \log (Z_5^2)$
6	$P_{12}(2)$	$P_{23}(3)$	$P_{31}(1)$	$P_{42}(2)$	Z_6	$-10 \log (Z_6^2)$
7	$P_{13}(3)$	$P_{21}(1)$	$P_{32}(2)$	$P_{42}(2)$	Z_7	$-10 \log (Z_7^2)$
8	$P_{13}(3)$	$P_{22}(2)$	$P_{31}(1)$	$P_{43}(3)$	Z_8	$-10 \log (Z_8^2)$
9	$P_{13}(3)$	$P_{23}(3)$	$P_{33}(3)$	$P_{41}(1)$	Z_9	$-10 \log (Z_9^2)$

Finalmente, são calculados os valores de S/N e selecionados os níveis com os valores de mais elevados, como pode ser analisado na tabela 23. É importante notar que a expressão 27 foi decomposta. Foram calculados parcialmente na tabela 22, onde foram calculados os rácios S/N das experiências e na tabela 23 e calculada a média para um nível.

Tabela 23 - Seleção dos Parâmetros

	P_1	P_2	P_3	P_4
1	$(S/N_1+S/N_2+S/N_3)/3$	$(S/N_1+S/N_4+S/N_7)/3$	$(S/N_1+S/N_6+S/N_8)/3$	$(S/N_1+S/N_5+S/N_9)/3$
2	$(S/N_4+S/N_5+S/N_6)/3$	$(S/N_2+S/N_5+S/N_8)/3$	$(S/N_3+S/N_5+S/N_7)/3$	$(S/N_2+S/N_6+S/N_7)/3$
3	$(S/N_7+S/N_8+S/N_9)/3$	$(S/N_3+S/N_6+S/N_9)/3$	$(S/N_2+S/N_4+S/N_9)/3$	$(S/N_3+S/N_4+S/N_8)/3$

É possível calcular a importância relativa dos parâmetros no desempenho da meta-heurística, através da variação do desempenho, dependendo dos valores do parâmetro, representados nas colunas da tabela 23. Os parâmetros que apresentem uma maior dispersão, têm maior impacto no desempenho da meta-heurística e devem ser analisados mais detalhadamente. Outros exemplos de planeamento de experiências de Taguchi na escolha dos parâmetros de meta-heurísticas pode ser analisado em [55-57].

3.8. HÍPER-HEURÍSTICAS

Embora o termo híper-heurísticas date do inicio do milénio, existem descrições anteriores de técnicas semelhantes. São descritas como mecanismos para encontrar, desenvolver ou parametrizar heurísticas para um problema de optimização. Burke et al. [58] apresenta uma definição formal: "*Métodos de pesquisa ou mecanismos de aprendizagem para selecionar ou gerar heurísticas para resolver problemas de optimização complexos*". O espaço de pesquisa de uma híper-heurística são heurísticas ou parâmetros de heurísticas, que pesquisam por soluções do problema [59,60]. Apresentam três características: são heurísticas de alto nível que controlam heurísticas de níveis mais baixos; são heurísticas que pesquisam por técnicas de otimização e não por soluções; utilizam apenas informação limitada do problema, permitindo serem utilizadas em problemas distintos [61]. Isto pode permitir desenvolver sistemas automáticos que independentemente do problema podem pesquisar por soluções sem qualquer intervenção do utilizador na escolha ou parametrização das meta-heurísticas utilizadas. Isto quer dizer que perante um problema de otimização, o sistema seleciona uma técnica de optimização ou mesmo uma sequência de técnicas de otimização, escolhe os parâmetros e resolve o problema [62,63].

Podem ser divididas entre híper-heurísticas de Seleção de Heurísticas, que determinam que heurísticas construtivas ou perturbativas serão utilizadas, e híper-heurísticas de Geração de Heurísticas, que constroem heurísticas através de componentes de heurísticas construtivas e perturbativas. Podem também ser divididas através do seu processo de discência, com híper-heurísticas que aprendem durante o processo de resolução do problema e híper-heurísticas que aprendem com experiências e posteriormente resolvem o problema. O processo de discência através da análise de experiências resulta no aumento do esforço computacional, pois é necessário fazer experiências para problemas novos [58].

Em [64] é apresentada uma híper-heurística que utiliza heurísticas sequencialmente na resolução do problema. É mantida uma lista tabu que não permite utilizar as heurísticas que não melhoraram a solução, enquanto a solução não melhorar. Em [65] é apresentado uma híper-heurística que seleciona em 12 heurísticas para escalonar um determinado número de tarefas. É utilizada um GA para selecionar as heurísticas e o tamanho do bloco de decisão, isto é, o número tarefas que devem ser escalonadas por uma determinada heurística numa iteração. Outra híper-heurística que apresenta um modelo semelhante é proposta em [66], com um bom desempenho.

3.9. CONCLUSÃO

Neste capítulo foram apresentados os conceitos fundamentais e o funcionamento básico das meta-heurísticas, depois de um breve enquadramento histórico, desde as primeiras heurísticas aos sistemas de apoio a decisão atuais. Posteriormente, foram apresentadas as técnicas de LS, que pesquisam numa determinada vizinhança por melhores soluções, e apresentado um exemplo ilustrativo. Foram ainda abordadas as formas de representação e os conceitos de vizinhança, necessários para compreender o funcionamento destas técnicas.

Um vez descritas as meta-heurísticas como mecanismos que orientam uma heurística subordinada na exploração do espaço de soluções para encontrar soluções próximas da ótima, foi apresentada a dicotomia entre os conceitos de intensidade e diversidade, que sustenta o funcionamento das meta-heurísticas, representando a relação entre eficiência e eficácia da pesquisa, que não deve ser muito restritiva ou aleatória. Foi ainda feita a distinção entre meta-heurísticas de solução única e meta-heurísticas de população, isto é, entre as baseadas na pesquisa de vizinhança e aquelas que utilizam uma população. Finalmente, foram analisadas seis meta-heurísticas, *Simulated Annealing* (SA), *Tabu Search* (TS), *Artificial Bee Colony* (ABC), *Ant Colony Otimization* (ACO), *Genetic Algorithms* (GA) e o *Particle Swarm Otimization* (PSO), apresentado o seu procedimento acompanhando por um exemplo ilustrativos do algoritmo das meta-heurísticas.

O capítulo terminou com uma análise das técnicas de parametrização de meta-heurísticas e apresentadas as técnicas de parametrização mais habituais. Em detalhe foi descrita a parametrização através do planeamento de experiências de Taguchi, que permite reduzir substancialmente o número de experiências necessárias e aumentar a eficiência. Foi ainda feita uma breve análise às híper-heurísticas que permitem parametrizar automaticamente, controlar e coordenar o funcionamento de heurísticas e meta-heurísticas.

4. IMPLEMENTAÇÃO E ANÁLISE DE RESULTADOS

Neste capítulo será apresentado o problema utilizado no estudo computacional e a implementação das meta-heurísticas selecionadas. Posteriormente, serão determinados os parâmetros das meta-heurísticas, pelo planeamento de experiências de Taguchi. Finalmente serão apresentados os resultados computacionais, estudadas detalhadamente as primeiras instâncias e será realizada uma análise do desempenho das meta-heurísticas. O capítulo termina com uma análise de desempenho das meta-heurísticas e a sua adequação à classe de problemas estudada, com base em inferência estatística.

4.1. PROBLEMA DE TESTE

Pretende-se no estudo computacional, analisar o desempenho técnicas de otimização no problema de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, utilizado no exemplo ilustrativo das meta-heurísticas do capítulo anterior. Este problema procura minimizar os atrasos ponderados, de um determinado número de tarefas, que devem ser executadas numa única máquina. Uma solução corresponde a uma sequência de execução das tarefas numa máquina, isto é, é necessário determinar uma permutação das tarefas que minimize os atrasos da ordem de fabrico, através da importância das tarefas.

Os modelos de sequenciamento, habitualmente, utilizam variáveis de decisão binárias que correspondem: à posição da tarefa na sequência de execução; ao momento em que a tarefa é ou começa à ser executada; à posição relativa da tarefa em relação a outras tarefa. Por exemplo $X_{jk} = 1$ se a tarefa *j* for colocada na posição *k*, $X_{jt} = 1$ se a tarefa *j* for executada no período *t* ou $X_{jt} = 1$ se a tarefa *j* começar a ser executada no momento *t* e $X_{ij} = 1$ se a tarefa *j* for precedida pela tarefa *i*. Embora existam modelos que usam outras variáveis de decisão estas são muito utilizadas em problemas onde é necessário determinar uma sequência de execução [67].

Em [65,66] são apresentados vários modelos de *IP* (*Integer Programming*) para problemas de sequenciamento e em [67] são apresentados modelos para o problema de minimização dos atrasos ponderados, $1||\Sigma w_j T_j$. Considerando uma variável que corresponde à posição da tarefa na sequência, o problema é modelado por [68,69]:

F.O.

$$Min \sum_{k=1}^{n} \sum_{j=1}^{n} w_j \, t_k \tag{28}$$

S.A.

$$\sum_{j=1}^{n} x_{jk} = 1, \qquad \forall \, k \tag{29}$$

$$\sum_{k=1}^{n} x_{jk} = 1, \qquad \forall \, i \tag{30}$$

$$\sum_{j=1}^{n} p_j \sum_{u=1}^{k} x_{ju} - \sum_{j=1}^{n} d_j x_{jk} \leq t_k, \qquad \forall \, k \tag{31}$$

$$t_k \geq 0, \qquad \forall \, k \tag{32}$$

Sendo t_k o atraso efetivo da tarefa na posição *k*, p_j o tempo de processamento da tarefa *j*, d_j a data de conclusão (*due date*) da tarefa *j,* w_j a importância da tarefa *j* e utilizando uma variável de decisão binária, onde $X_{jk} = 1$ se a tarefa *j* for colocada na posição *k*, a F.O. de minimização dos atrasos ponderados é apresentada em (28).

O problema é constituído por 4 restrições. As restrições 29 e 30, impõem limitações na sequenciação. Em 29, é definido que uma tarefa apenas pode ocupar uma posição da sequência e em 30, é definido que uma posição apenas pode ser ocupada por uma tarefa, isto é, não podem existir duas tarefa alocadas na mesma posição e uma tarefa não pode ocupar varias posições simultaneamente. Em 31, é definido o atraso na tarefa na posição *k*, calculado através da subtração do tempo de execução da tarefa e das tarefa que a antecedem, e a data de conclusão. Finalmente, a restrição 32, serve para considerar o atraso efetivo, que é equivalente ao $max(L_j, 0)$.

Outros modelos para o problema de minimização dos atrasos ponderados podem ser analisados em [69]. O autor conclui que o modelo utilizado não é o mais habitual embora apresente benefícios. Em [68] são apresentados os benefícios deste modelo, sendo mais apropriado para técnicas exatas aplicadas por utilizadores mais experientes.

O problema de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, além do interesse académico, como problema de sequenciamento, é bastante importante em contextos industriais. O mercado extremamente competitivo, tornou absolutamente necessário cumprir os prazos estipulados. Perante a impossibilidade de cumprir os prazos é necessário fazer uma ponderação da importância das encomendas e dos clientes, o que vai permitir à empresa determinar que encomendas implicam uma maior penalização quando os prazos não são cumpridos. É assim, necessário escolher uma sequência de fabrico para minimizar os atrasos ponderados pela importância da encomenda ou clientes.

4.2. INSTÂNCIAS DE TESTE

O estudo computacional será realizado com 30 instâncias de 50 tarefa, disponíveis no *ORLibrary* [70]. O recurso às instâncias de problemas académicos do *ORLibrary* permite conhecer os resultados ótimos, facilitando a análise estatística do resultados, pois torna possível comparar instâncias diferentes, através do desvio em relação ao ótimo e ainda comparar como os resultados obtidos por outros autores com resultados publicados.

O *ORLibrary* disponibiliza 125 instâncias para o problemas de minimização dos atrasos efetivos ponderados, com 40, 50 e 100 tarefas, com tempos de execução uniformemente distribuídos entre [0,100] e pesos entre [0,10]. Foram escolhidas instâncias com 50 tarefas o que permite analisar o desempenho das meta-heurísticas em problemas complexos. O sequenciamento de 50 tarefas corresponde a 3.04×10^{64} (*50!*) soluções.

4.3. IMPLANTAÇÃO DO PROTÓTIPO

O estudo computacional vai avaliar e comparar o desempenho de duas meta-heurísticas. Foi selecionado o SA, que é uma meta-heurística de solução única muito eficiente, e o DABC, que é uma adaptação do ABC para problemas discretos com resultados promissores [41-44]. O DABC é relativamente recente e pretende-se com esta análise fazer uma comparação com uma meta-heurística com um desempenho e eficiência comprovada. Tanto o SA como o DABC foram implementadas em *C*, no Microsoft Visual Studio 2013. Foi adoptada uma implementação modular, onde cada meta-heurística é um módulo. Os outros módulos foram implementados para serem utilizados por ambas as meta-heurísticas Este tipo de implementação permite adaptar rapidamente o funcionamento do protótipo, por exemplo, para outros problemas, introduzindo um novo módulo de avaliação das soluções. Uma representação do protótipo poderá ser analisado na figura 19.

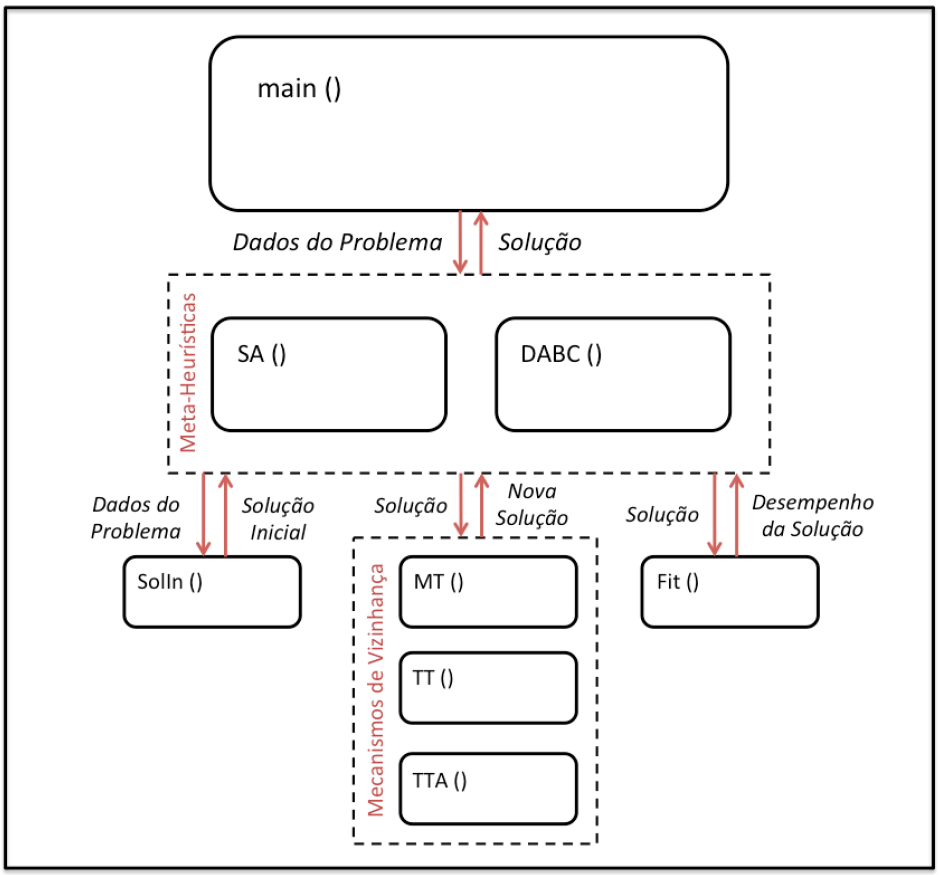

Figura 19 - Arquitetura do Protótipo

Como é possível analisar, no esquema da arquitetura, existem 7 módulos diferentes. O SA e o DABC executam o procedimento das meta-heurísticas, descritos nas tabelas 8 e 14 respectivamente, com recurso aos módulos de formação da solução inicial, de avaliação de desempenho e dos mecanismos de vizinhança implementados.

Os módulos implementados podem ser divididos em módulos de meta-heurísticas, que são utilizados pelo *main ( )*, um módulo que desenvolve soluções aleatórias, três módulos de pesquisa de vizinhanças e um módulo de avaliação do desempenho, que são utilizados pelas meta-heurísticas. O procedimento de cada módulo pode ser descrito por:

- ***main ( )*** - Onde são recolhidos os dados do problema e imprimidos os resultados num ficheiro .txt. É também no *main ( )* que é medido o tempo de execução de cada meta-heurística, que também é apresentado no ficheiro .txt.
- ***SA ( )*** - Onde é executado o *SA*, isto é, depois de conhecidos os dados do problema e os parâmetros da meta-heurística é feita uma pesquisa pelo espaço de soluções através do procedimento descrito na tabela 8.
- ***DABC ( )*** - Onde é executado o *DABC*, isto é, depois de conhecidos os dados do problema e os parâmetros da meta-heurística é feita uma pesquisa pelo espaço de soluções através do procedimento descrito na tabela 14.
- ***SolIn ( )*** - Onde é gerada uma sequência de tarefas aleatória, que são utilizados como soluções iniciais do SA e DABC, bem como pelas abelhas exploradoras do DABC para encontrar novas fontes de alimentos.
- ***MT ( )*** – Corresponde ao módulo geração de vizinhanças, isto é, uma forma de transformar uma sequência de tarefas numa outra, através de movimento de uma tarefa para outra posição aleatória da sequência.
- ***TT ( )*** - Corresponde ao módulo geração de vizinhanças, isto é, uma forma de transformar uma sequência de tarefas numa outra, através da troca de posição de duas tarefas, escolhidas aleatoriamente.
- ***TTA ( )*** - Corresponde ao módulo geração de vizinhanças, isto é, uma forma de transformar uma sequência de tarefas numa outra, através da troca de posição de duas tarefas adjacentes, escolhidas aleatoriamente.
- ***Fit ( )*** - Onde, conhecidos os dados do problema, é calculado o desempenho da solução. Para o problema em questão é calculado o atraso efetivo ponderado para uma determinada de sequência de execução.

Esta implementação modular permite flexibilizar a estrutura do protótipo, incorporando novas meta-heurísticas e outros módulos para controlar a pesquisa e avaliar as soluções. Isto quer dizer que é fácil introduzir um novo mecanismo de vizinhança ou utilizar as meta-heurísticas noutros problemas de sequenciamento, incorporando novos módulos.

4.4. PARAMETRIZAÇÃO DO SA

Os parâmetros do SA têm muito impacto no desempenho da meta-heurística. Uma boa parametrização resulta numa pesquisa eficiente que permite encontrar boas soluções. O impacto dos parâmetros apenas é considerável se não houver muito tempo para executar o SA, isto é, se for considerada uma temperatura inicial alta que decresce lentamente, a meta-heurística vai encontrar boas soluções, mesmo que de forma menos eficiente [71]. Os parâmetros principais do SA são: a temperatura inicial; o mecanismo de vizinhança; o número de iterações à mesma temperatura; o mecanismo de arrefecimento; o critério de interrupção. O número de iterações foi definido em 10000, os restantes parâmetros foram determinados utilizando o 1ª instância do estudo computacional, que é semelhante as outras. Os valores dos parâmetros foram determinados por:

- ***Temperatura Inicial (T_i)*** - Em [33] é indicado que a temperatura inicial deverá resultar numa probabilidade de aceitação de soluções piores próxima de 1, isto é, no inicio o SA, deverá aceitar quase todas as soluções. Para determinar a temperatura inicial é possível utilizar a probabilidade de aceitação. Em [35] é indicada uma probabilidade de aceitação de soluções piores [0.7,0.8] para calcular a temperatura inicial através da expressão 33, onde P_i representa a probabilidade de aceitação de soluções piores e $\Delta f(x)$ a dispersão da amostra [72,73].

$$T_i = \frac{\Delta f(x)}{\ln P_i} \tag{33}$$

Foram retiradas 20 soluções aleatórias para calcular a dispersão média e consideradas probabilidades de aceitação de 0.70, 0.75 e 0.80, resultando em temperaturas iniciais, arredondadas de, 750, 1000 e 1250, respectivamente. Posteriormente será determinada a temperatura inicial, recorrendo ao planeamento de experiências de Taguchi.

- ***Mecanismos de Vizinhança*** - Em [71] é indicada a importância dos mecanismos de vizinhança no desempenho do SA. Os mecanismos de vizinhança escolhidos devem permitir fazem um balanço entre a intensidade e a diversidade da pesquisa da meta-heurística. Em [35] é utilizado um esquema de vizinhança de troca de pares de tarefas adjacentes, para problemas de minimização dos atrasos efetivos ponderados, com 50 tarefas, com resultados próximos do ótimo. Outros mecanismos do vizinhança do SA podem ser analisados em [74].

Foram implementados 3 esquemas de vizinhança. O movimento de uma tarefa (MA) determinada aleatoriamente para uma nova posição, também aleatória, a troca de posição de um par de tarefa aleatórias (TA) e a troca de posições de um par de tarefas adjacentes (TTA), determinadas aleatoriamente.

- ***Iterações à Mesma Temperatura* (*L*)** - Em [71] é indicado que o número de iterações à mesma temperatura deve estar próximo do tamanho do problema, por exemplo o número de tarefas num problema de sequenciamento ou o número de cidades no problema do *Traveling Salesman Problem*. O número de iterações também poderá depender do mecanismo de vizinhança, isto é, ser idêntico ao número de vizinhos. Por exemplo no problema da tabela 9 cada solução apresenta 6 vizinhos, sendo aconselhável 6 iterações à mesma temperatura.

A instância utlizada tem 50 tarefas e tendo em consideração que os mecanismos de vizinhança podem resultar num número de vizinhos superior ao número de tarefas foram escolhidas 50, 75 ou 100 iterações à mesma temperatura Posteriormente será determinado *L*, recorrendo ao planeamento de experiências de Taguchi.

- ***Mecanismo de Arrefecimento* (*α*)** - Em [12] são apresentados vários esquemas de arrefecimento, incluindo o arrefecimento proporcional (geométrico), que consiste na multiplicação da temperatura por uma constante (α). Em [73] é indicado que o arrefecimento deverá ser executado lentamente, com valores de α entre [0.80,0.99]. Já em [71] é apresentada uma forma de calcular α, que pode ser analisada na expressão 34, onde T_i representa a temperatura inicial, T_f representa a temperatura final escolhida e M representa o número de patamares de temperatura.

$$\alpha = \left(T_f / T_i\right)^{1/(M-1)} \tag{34}$$

Considerando as temperaturas iniciais (750, 1000 e 1250), o número de iterações à mesma temperatura (50, 75 e 100), o numero de iterações (10000) e temperatura final sempre maior que 1, os valores de α devem ser 0.97, 0.98 ou 0.99. Foram utilizados os menores valores dos outros parâmetros (750 e 50) para determinar α.

Posteriormente serão determinados os valores específicos dos parâmetros através do planeamento de experiências de Taguchi com 4 parâmetros e 3 níveis.

4.4.1. ESCOLHA DOS PARÂMETROS DO SA

Os parâmetros determinados serão analisados utilizando as experiências de Taguchi,. Na tabela 24 é possível analisar os níveis dos parâmetros, respectivamente, temperatura inicial, mecanismos de vizinhança, número de iterações à mesma temperatura e mecanismo de arrefecimento. Pretende-se encontrar a combinação com melhor desempenho, isto é, que resulta no comportamento mais eficaz do SA. Foram utilizadas 10000 iterações, um número muito reduzido, particularmente quando comparado as 3.04×10^{64} soluções.

Tabela 24 - Parâmetros do SA

	1	**2**	**3**
P_1	*750*	*1000*	*1250*
P_2	*MA*	*TA*	*TAC*
P_3	*50*	*75*	*100*
P_4	*0.97*	*0.98*	*0.99*

Uma vez determinados os valores dos parâmetros, podem ser realizadas as experiências, utilizando o primeiro problema do estudo computacional. Para 4 parâmetros e 3 níveis é possível recorrer à matriz L_9, que pode ser analisado na tabela 25. Uma experiência factorial completa destas características necessitava de 81 experiências, enquanto o planeamento de experiências de Taguchi apenas necessita de 9, sendo muito mais eficiente. Os resultados das 9 experiências são representados em *f(x)*.

Tabela 25 - S/N das Experiências do SA

	P_1	***P_2***	***P_3***	***P_4***	***f(x)***	***S/N***
1	*750*	*MA*	*50*	*0.97*	*2273*	*-67.132*
2	*750*	*TA*	*100*	*0.98*	*2213*	*-66.900*
3	*750*	*TAC*	*75*	*0.99*	*3643*	*-71.299*
4	*1000*	*MA*	*100*	*0.99*	*2651*	*-68.468*
5	*1000*	*TA*	*75*	*0.97*	*2184*	*-66.785*
6	*1000*	*TAC*	*50*	*0.98*	*4347*	*-72.764*
7	*1250*	*MA*	*75*	*0.98*	*2314*	*-67.287*
8	*1250*	*TA*	*50*	*0.99*	*2334*	*-67.362*
9	*1250*	*TAC*	*100*	*0.97*	*3043*	*-69.666*

Finalmente, é possível calcular os valores S/N dos parâmetros, que podem ser analisados na tabela 26. Os níveis de um parâmetro que apresentam os valores de S/N mais elevados, encontram as melhores soluções e serão utilizados no estudo computacional.

Tabela 26 - S/N dos Parâmetros do SA

	P_1	P_2	P_3	P_4
1	*-68.444*	*-67.629*	*-69.086*	*-67.861*
2	*-69.339*	*-67.016*	*-68.457*	*-68.457*
3	*-68.105*	*-71.243*	*-68.345*	*-69.043*

Os valores de S/N por parâmetros podem ser analisados na fig. 20. O estudo computacional vai utilizar uma temperatura inicial de 1250, o mecanismo de vizinhança de troca de tarefas (TT), um numero de iterações à mesma temperatura de 100 e um mecanismo de arrefecimento (α) de 0.97. Na figura 20 também é possível verificar a importância relativa dos parâmetros no desempenho da meta-heurística. Os parâmetros que apresentem uma maior dispersão dos valores de S/N, apresentam maior impacto no desempenho e devem por isso ser alvo de atenção especial. O mecanismo de vizinhança é o parâmetro com maior impacto no desempenho do SA, devido ao mau desempenho da troca de tarefas adjacentes (TTA), que resultou nas piores soluções da experiência. Os restantes parâmetros resultaram numa variação bastante mais reduzida do desempenho do SA.

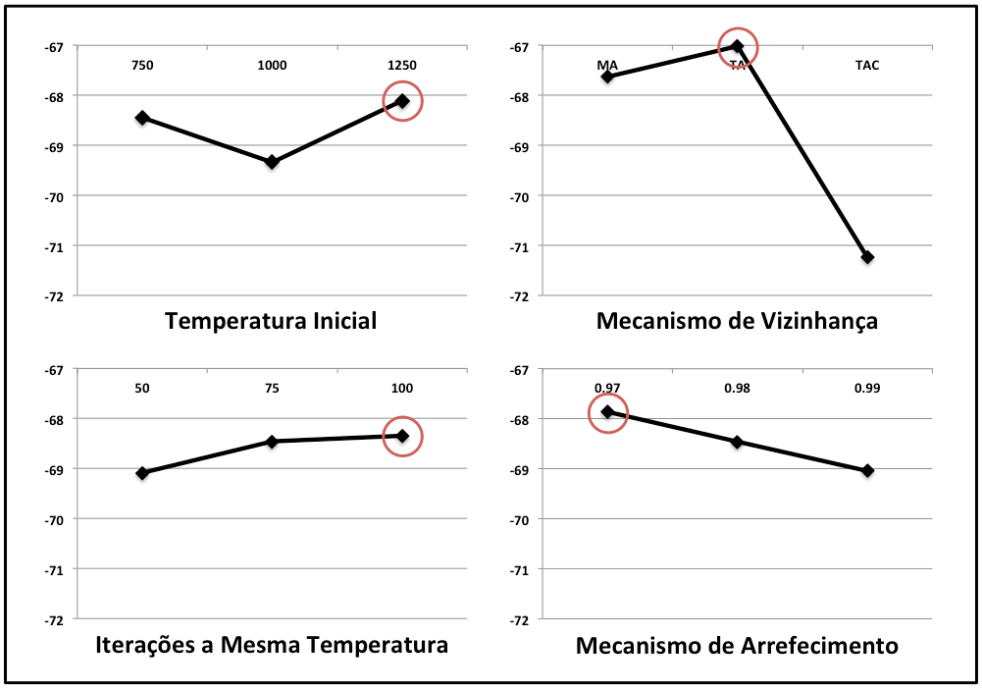

Figura 20 - Seleção dos Parâmetros do SA

4.5. PARAMETRIZAÇÃO DO DABC

Os parâmetros do DABC têm um enorme impacto no desempenho da meta-heurística, uma boa parametrização resulta numa pesquisa eficiente, isto é, uma pesquisa suficientemente intensa e diversificada. O ABC, e consequentemente o DABC, são meta-heurísticas com poucos parâmetros [75], no entanto, no estudo computacional será considerado um parâmetro que não é habitualmente utilizado e que poderá ter impacto nas características da pesquisa. Os parâmetros genéricos do ABC são: o tamanho da colónia; a proporção de abelhas; o mecanismos de vizinhança; o número limite; o critério de interrupção. Existem outros parâmetros que podiam ser utilizados, que no entanto não foram considerados na implementação. O número de iterações foi definido em 1000. Foi escolhido um número de iterações menor que no SA porque serão exploradas múltiplas soluções por iteração Os parâmetros foram determinados na 1ª instância, considerando que todas as instâncias tem características semelhantes. Os valores dos parâmetros foram determinados por:

- ***Tamanho da Colónia* (*L*)** - O tamanho da colónia vai indicar o número de soluções exploradas em cada iteração, isto é, colónias maiores resultam em pesquisas mais pesadas e minuciosas. Em [76] é realizado um estudo sobre o impacto dos parâmetros no ABC, utilizando colónias semelhante ao tamanho do problema. O estudo computacional demostra que o tamanho da colónia não necessita ser determinado minuciosamente. Em [77] é apresentado outro estudo sobre o impacto dos parâmetros, resultando numa colónia de 50 indivíduos.

As instâncias do estudo computacional têm 50 tarefas e foram utilizados tamanhos de colónia de 10, 20 e 30. O facto do desempenho da meta-heurística não depender muito do tamanho da colónia levou a não escolher colónias com mais de 30 indivíduos. O tamanho da colónia será determinado recorrendo ao planeamento de experiências de Taguchi.

- ***Proporção de Abelhas*** - Habitualmente a proporção entre abelhas oportunistas e trabalhadoras/exploradoras é fixada em 50%, isto é, metade das abelhas são abelhas oportunistas e metade trabalhadoras/exploradoras. Uma alteração da proporção pode ter muito impacto no desempenho da meta-heurística. O DABC com mais abelhas oportunistas faz pesquisas com maior intensidade, enquanto mais trabalhadoras/exploradoras resulta em maior diversidade. Este parâmetro apresenta características semelhantes à fixação de um número de abelhas exploradoras.

Foram consideradas proporções de abelhas de 40/60, 50/50 e 60/40 da colónia, isto é, 40%, 50% ou 60% das abelhas da colónia são abelhas oportunistas e as restantes trabalhadoras/exploradoras. O ABC habitualmente utiliza proporções de 50/50, sendo escolhidos valores próximos, que alterem pouco as características da meta-heurística.

- ***Mecanismos de Vizinhança*** – Os mecanismos de vizinhança vão controlar a pesquisa e devem permitir fazem uma balanço entre a intensidade e a diversidade. O número de vizinhos e alterações numa iteração são particularmente importantes, porque determinam os movimentos das abelhas. Mecanismos de vizinhança que apenas alteram uma pequena parte da solução por iteração vão deslocar-se muito lentamente, resultando numa pesquisa intensa e pesada, que poderá ser muito pouco eficiente. Outro mecanismo de vizinhança pode ser analisado em [44].

Foram implementados os mesmos mecanismos de vizinhança utilizados no SA. O movimento de uma tarefas (MT), para uma nova posição selecionada aleatoriamente, a troca de posição de um par de tarefas selecionada aleatoriamente (TT), e a troca de posições de um par de tarefas adjacentes (TTA), determinados aleatoriamente.

- ***Número Limite (l)*** – O impacto do desempenho do número limite está relacionado com o tamanho da colónia, isto é, colónias maiores não são afectadas por números limites pequenos [76,78]. O impacto reduzido do número limite é explicado pela diversidade resultante de colónias maiores. Em [44] é apresentada a expressão 35, para calcular o número limite, onde *cs* representa o tamanho da colónia e *D* a dimensão do problema. Outras formas de calcular o número limite podem ser analisadas em [76,78], semelhantes à apresentada em 35.

$$l = \frac{cs \times D}{3} \tag{35}$$

Os problemas do estudo computacional têm 50 tarefas e utilizando colónias de 10, 20 e 30, foram escolhidos números limite de 500, 550 e 600, o que irá permitir fazer uma pesquisa suficientemente intensa das fontes de alimento antes de serem consideradas exaustas pela colónia e serem trocadas por novas fontes de alimento.

Posteriormente serão determinados os valores específicos dos parâmetros através do planeamento de experiências de Taguchi com 4 parâmetros e 3 níveis.

4.5.1. ESCOLHA DOS PARÂMETROS DO DABC

Os parâmetros determinados serão analisados utilizando as experiências de Taguchi. Na tabela 27 é possível analisar os níveis dos parâmetros, respectivamente, o tamanho da colónia, a proporção de abelhas, o mecanismos de vizinhança e o número limite. Pretende-se encontrar a combinação com melhor desempenho, isto é, que resulta no comportamento mais eficaz do DABC. O número de iterações é 1000, que será também utilizado no estudo computacional e é cerca de 10 vezes menor que o utilizado pelo SA.

Tabela 27 - Parâmetros do DABC

	1	**2**	**3**
P_1	*10*	*20*	*30*
P_2	*MA*	*TA*	*TAC*
P_3	*40/60*	*50/50*	*60/40*
P_4	*500*	*550*	*600*

Uma vez determinados os valores dos parâmetros serão realizadas as experiências, recorrendo ao primeiro problema do estudo computacional. Para 4 parâmetros e 3 níveis é utilizada a matriz L_9, que pode ser analisada na tabela 28. Uma experiência factorial completa destas características necessitava de nove vezes mais experiências que aquelas necessárias no planeamento de experiências de Taguchi, que é bastante mais eficiente. Os resultados das 9 experiências são representados em *f(x)*.

Tabela 28 - S/N das Experiências do DABC

	P_1	***P_2***	***P_3***	***P_4***	***f(x)***	***S/N***
1	*10*	*MA*	*40/60*	*500*	*2184*	*-66.785*
2	*10*	*TA*	*60/40*	*550*	*2134*	*-66.584*
3	*10*	*TAC*	*50/50*	*600*	*3988*	*-72.015*
4	*20*	*MA*	*60/40*	*600*	*2273*	*-67.132*
5	*20*	*TA*	*50/50*	*500*	*2184*	*-66.785*
6	*20*	*TAC*	*40/60*	*550*	*4493*	*-72.051*
7	*30*	*MA*	*50/50*	*550*	*2184*	*-66.785*
8	*30*	*TA*	*40/60*	*600*	*2163*	*-66.701*
9	*30*	*TAC*	*60/40*	*500*	*4749*	*-73.532*

É possível calcular os valores S/N dos parâmetros, que podem ser analisados na tabela 29. Os níveis de um parâmetro que apresentam os valores de S/N mais elevados, revelaram ser mais eficazes e serão utilizados no estudo computacional.

Tabela 29 - S/N dos Parâmetros do DABC

	P_1	P_2	P_3	P_4
1	*-68.461*	*-66.901*	*-68.846*	*-69.034*
2	*-68.989*	*-66.690*	*-68.528*	*-68.807*
3	*-69.006*	*-72.866*	*-69.083*	*-68.616*

Os valores de S/N podem ser examinados na tabela 29. No estudo computacional vão ser utilizadas 10 abelhas, o mecanismo de vizinhança de troca de tarefas (TT), um proporção entre abelhas oportunistas e trabalhadoras/exploradoras de 50/50 e um número limite (*l*) de 600. Na figura 21 também é possível verificar a importância relativa dos parâmetros no desempenho da meta-heurística. Os parâmetros que apresentem uma maior dispersão dos valores de S/N, têm mais impacto no desempenho e devem por isso ser alvo de atenção especial. O mecanismo de vizinhança é o parâmetro com maior impacto do desempenho do DABC, devido ao mau desempenho da troca de tarefas adjacentes (TTA), que resultou nas piores soluções da experiência. Os restantes parâmetros resultaram numa variação mais reduzida do desempenho da meta-heurística.

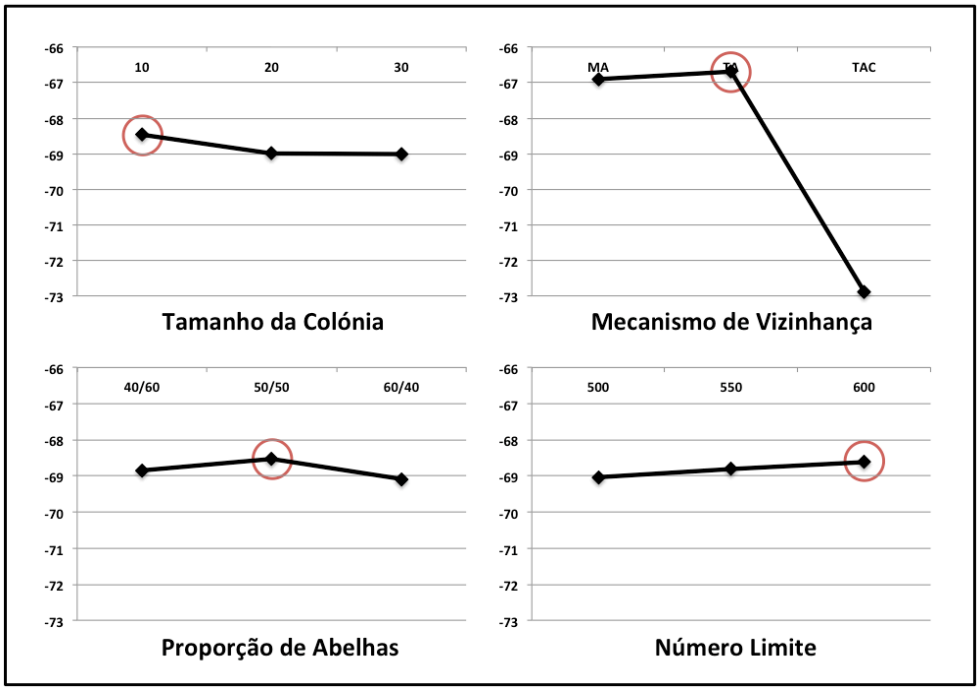

Figura 21 - Seleção dos Parâmetros do DABC

4.6. RESULTADOS COMPUTACIONAIS

O estudo computacional foi realizado em 3 repetições para cada instância, sendo apenas considerado o melhor desempenho de cada meta-heurística, num MacBook Pro (Retina), com um processador 3GHz Intel Core i7, 16GB 1600 MHz DDR3 de memória e utilizando o Windows 8. Os resultados das instâncias de 1 ao 15 são apresentados na tabela 30, onde é possível analisar o resultado das repetições do SA e DABC e a solução ótima de cada instância disponíveis no *ORLibrary* [68].

Tabela 30 - Resultados das Instâncias *wt1* a *wt15*

	SA			***DABC***			***Sol. Ótima***
	1	***2***	***3***	***1***	***2***	***3***	
wt1	2235	2247	2305	2134	2134	2184	2134
wt2	2237	2159	2096	1998	2011	1998	1996
wt3	2583	2612	2583	2583	2715	2619	2583
wt4	2691	3049	2691	2909	2691	2765	2691
wt5	1657	1604	1762	1522	1604	1518	1518
wt6	26863	26733	26885	26583	26327	26296	26276
wt7	11689	11767	11583	11534	11554	11816	11403
wt8	8867	8758	8856	8766	8668	8582	8499
wt9	9979	9957	9925	9884	10072	9931	9884
wt10	10863	10772	10823	10655	10862	10943	10655
wt11	44012	44156	44425	43733	43921	43620	43504
wt12	37384	36904	37017	36755	36778	36612	36378
wt13	45922	46211	45877	45725	45555	45838	45383
wt14	52140	52227	52468	52150	52095	52341	51785
wt15	39094	39615	39142	39087	38974	39049	38934

Foram realizadas 10000 iterações no SA e 1000 no DABC, que analisa 10 soluções por iteração, resultando em 10000 soluções analisadas pelo DABC. Os restantes parâmetros das meta-heurísticas foram os determinados anteriormente, nomeadamente T_i de 1250, o mecanismo de vizinhança de troca de tarefas (TT), L de 100 e α de 0.97 no SA e uma colónia de 10 abelhas, o mecanismo de vizinhança de TT, um proporção entre abelhas oportunistas e trabalhadoras/exploradoras 50/50 e l de 600 no DABC.

Os resultados dos problemas de 16 ao 30 são apresentados na tabela 31, onde é possível analisar o resultado das corridas do SA e DABC e a solução ótima de cada problema disponíveis no *ORLibrary* [68]. Embora as instâncias do estudo computacional tenham tempos de execução uniformemente distribuídos entre [0,100] e pesos entre [0,10], isto é, todas as instâncias têm características semelhantes, as datas de conclusão variam entre as instâncias, como e possível constar nas soluções ótimas.

Tabela 31 - Resultados das Instâncias *wt16* a *wt30*

	SA			***DABC***			***Sol. Ótima***
	1	***2***	***3***	***1***	***2***	***3***	
wt16	88760	88437	88694	88730	88389	88568	87902
wt17	84911	85656	84921	84580	84338	84499	84260
wt18	106245	106330	106175	105146	105406	105604	104795
wt19	90073	90193	89779	89886	89694	89771	89299
wt20	73061	73497	73880	72730	72751	72559	72316
wt21	215358	215340	215466	214983	214884	214702	214546
wt22	151900	151561	151586	150966	151207	151115	150800
wt23	224832	224582	224918	224621	224399	224476	224025
wt24	117090	116980	116745	116164	116418	116326	116015
wt25	240959	240748	240957	240538	240652	240641	240179
wt26	10	78	2	24	2	2	2
wt27	4	4	4	4	4	4	4
wt28	852	1112	806	811	755	815	755
wt29	214	104	104	99	226	104	99
wt30	22	22	22	95	22	22	22

Os resultados, apresentados nas tabelas 27 e 28, permitem demostrar o desempenho do SA e DABC. Ambas as meta-heurísticas apresentaram um desempenho muito próximo do ótimo. O SA encontrou a solução ótima em 5 instâncias, 16.666% das instâncias, enquanto o DABC encontrou a solução ótima em 11 instâncias, 36.666% das instâncias. Embora o SA não tenha encontrado nenhuma solução ótima que o DABC não tenha encontrado, para retirar conclusões acerca do desempenho do SA e DABC é necessário avaliar o desvio médio das soluções não ótimas de ambas as técnicas.

Os tempo de processamento das meta-heurísticas podem ser analisados na tabela 32, sendo apenas apresentados os tempos de processamento da melhor corrida. É importante notar que os valores apresentados têm uma componente de incerteza associada, pois foram medidos em ciclos de processador e posteriormente transformados em segundos. O SA, habitualmente, necessita de menos tempo computacional que o DABC, mesmo tendo em consideração as 10000 iterações do SA comparadas com as 1000 do DABC, tal deve-se ao procedimento do DABC que explora várias soluções em cada iteração.

Tabela 32 - Tempos de Processamento

Instância	***1***	***2***	***3***	***4***	***5***	***6***
SA	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
DABC	< 0.001	< 0.001	< 0.001	0.015	< 0.001	0.016
Instância	***7***	***8***	***9***	***10***	***11***	***12***
SA	0.015	< 0.001	< 0.001	< 0.001	0.031	< 0.001
DABC	0.031	0.016	0.016	0.016	0.016	0.016
Instância	***13***	***14***	***15***	***16***	***17***	***18***
SA	0.031	< 0.001	< 0.001	< 0.001	0.031	< 0.001
DABC	0.015	0.015	0.031	0.015	0.031	0.016
Instância	***19***	***20***	***21***	***22***	***23***	***24***
SA	0.031	< 0.001	0.031	< 0.001	< 0.001	< 0.001
DABC	0.016	0.031	0.016	0.016	0.016	0.032
Instância	***25***	***26***	***27***	***28***	***29***	***30***
SA	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
DABC	0.015	< 0.001	0.015	< 0.001	0.031	< 0.001

Os resultados apresentados na tabelas 29, demostram a eficiência do SA e DABC, sendo capazes de resolver o problema de uma forma praticamente instantânea. É impossível contabilizar os tempos de processamento do SA na maioria dos problemas, pois é menor que 0.001s. Embora o DABC demore mais tempo, o tempo de processamento contínua a ser pouco significativo, sendo sempre inferior que 0.032s. O tempo de processamento é afectado pelas características do problema e pela componente aleatória das meta-heurísticas, resultando em tempos de processamento diferentes da mesma meta-heurística numa instância, pois pode ser necessário fazer mais ou menos cálculos.

O desempenho do SA e DABC pode ser comparado na figura 22. Como é possível constatar ambas encontraram soluções ótimas ou muito próximas da solução ótima. O DABC obteve um desempenho, quase sempre, superior ao SA.

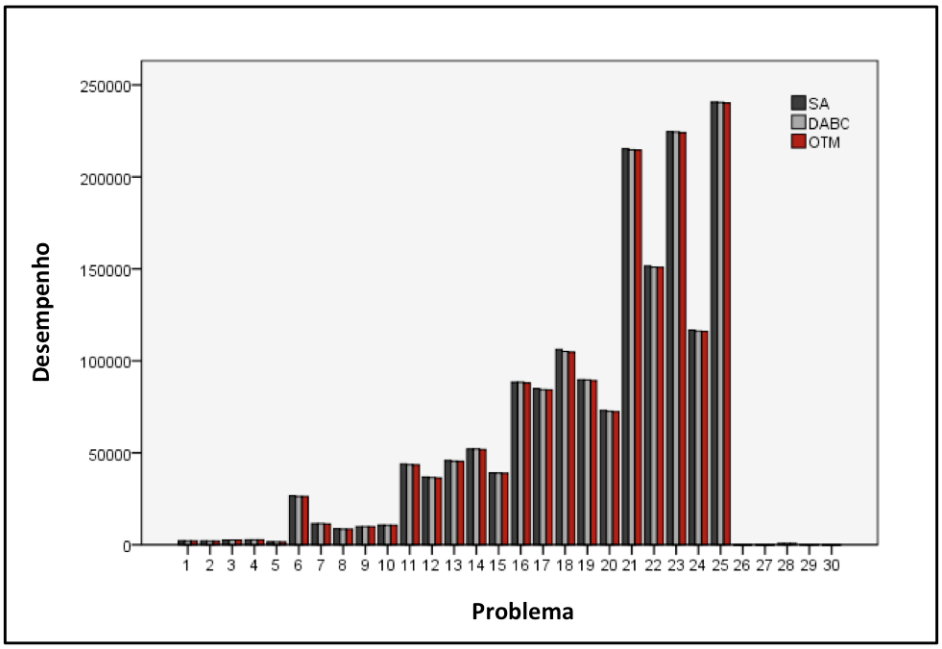

Figura 22 - Desempenho do SA e DABC

4.6.1. ANÁLISE DA INSTÂNCIA *WT1*

O SA encontrou uma solução de 2235 na instância *wt1*, com um desvio de 101 (4.733%) da solução ótima. Na figura 23 é possível analisar o funcionamento do SA. Durante as primeiras 94 iterações melhora rapidamente e posteriormente paralisa. Na maioria das instâncias do estudo computacional são necessarias bastantes mais iterações.

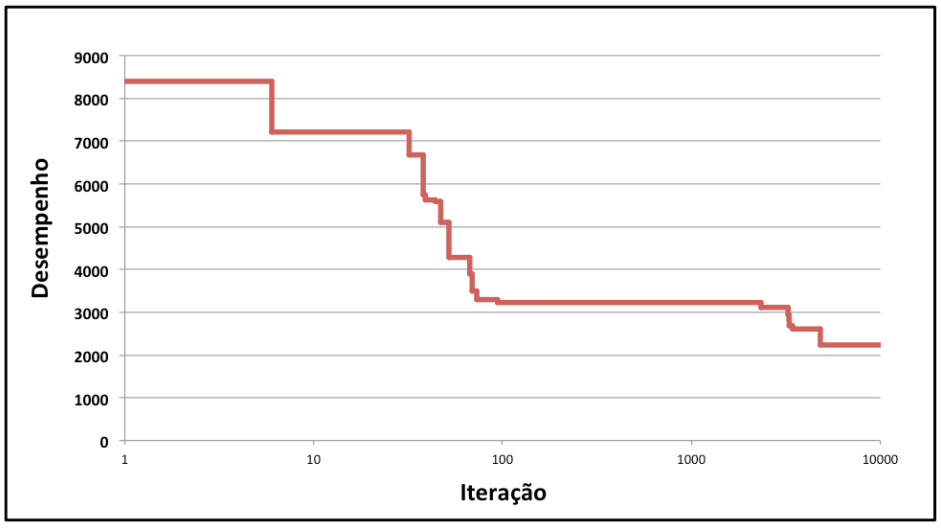

Figura 23 - Evolução do SA na Instância *wt1*

Na figura 24 é possível analisar o comportamento do DABC na resolução da instância *wt1*, que encontrou a solução ótima em 589 iterações. O DABC encontra muito rapidamente melhores soluções, particularmente nas primeiras 64 iterações, posteriormente são necessárias mais iterações para encontrar melhores soluções.

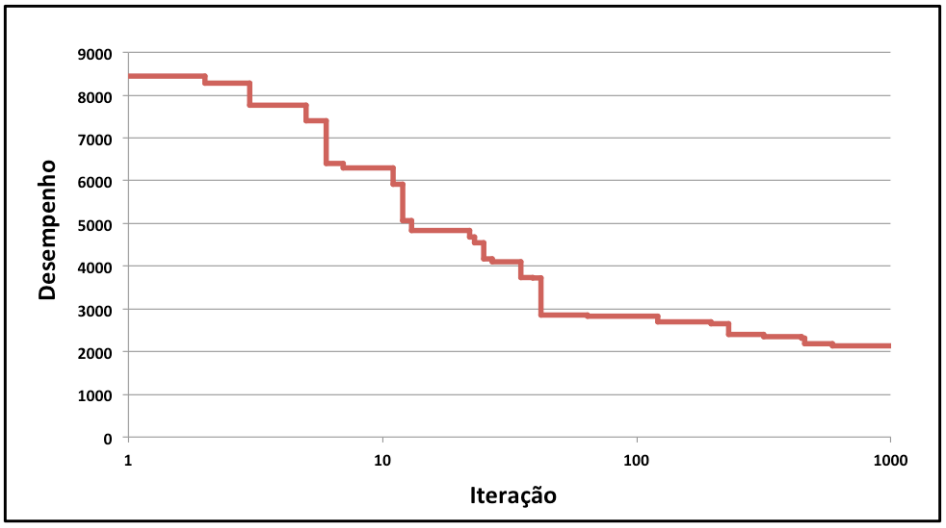

Figura 24 - Evolução do DABC na Instância *wt1*

4.6.2. ANÁLISE NA INSTÂNCIA *WT2*

O SA encontrou uma solução de 2096, com um desvio de 101 (5.010%) da solução ótima, em 7755 iterações, como é possível analisar na figura 25. Tal como acontece na instância *wt1*, na instância *wt2*, o SA também melhora muito rapidamente a solução, particularmente desde da solução inicial de 25558 até à iteração 328 onde é encontrada uma solução com desempenho 5343. Posteriormente o processo de melhoria da solução é mais lento.

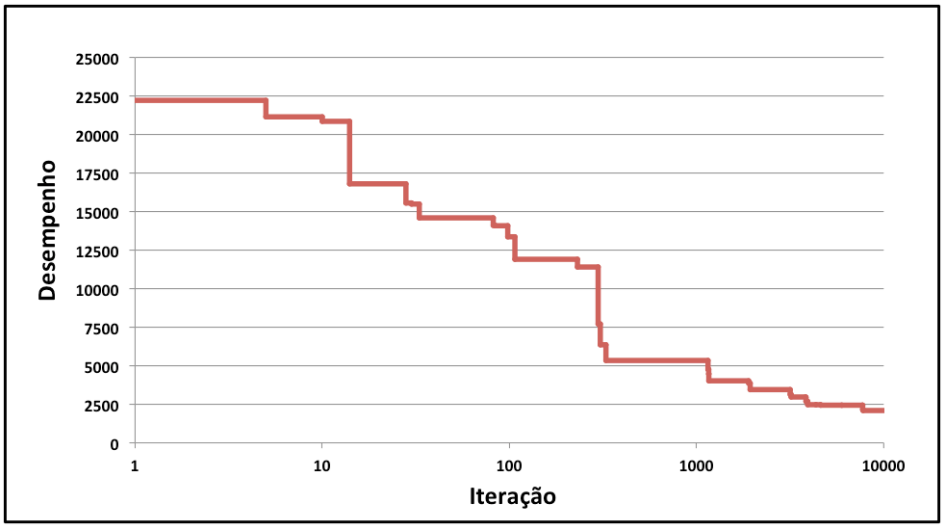

Figura 25 - Evolução do SA na Instância *wt2*

O DABC encontrou uma solução muito próxima da ótima, 1998 com um desvio de 2 (0.100%) do ótimo de 1996, como é possível analisar na figura 26. São necessárias 808 iterações para encontrar a solução, depois de manter a solução 2025 por 463 iterações. Um pequeno aumento do número de iterações poderia permitir encontrar a solução ótima.

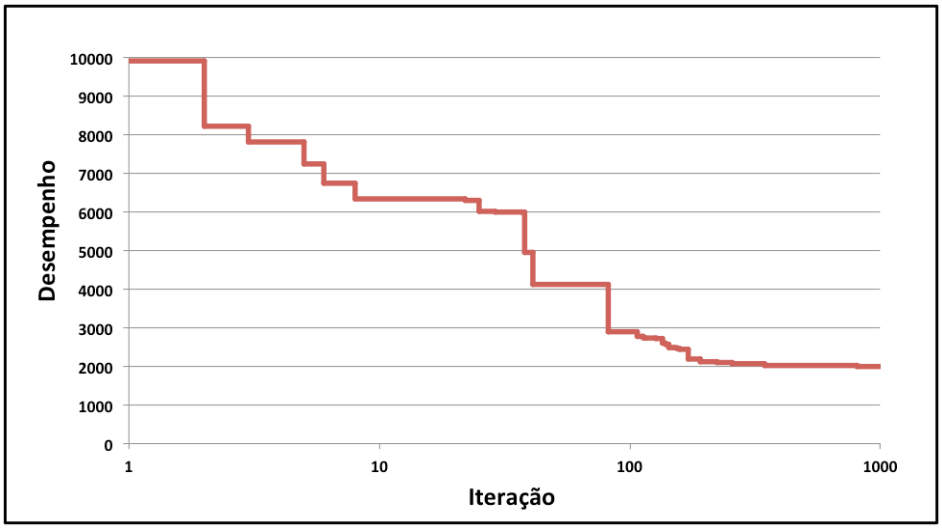

Figura 26 - Evolução do DABC na Instância *wt2*

4.6.3. ANÁLISE NA INSTÂNCIA *WT3*

Na instância *wt3* o SA encontrou a solução ótima, de 2583, em 6248 iterações, como pode ser analisado na figura 27. Durante as primeiras 803 iterações as soluções são melhoradas rapidamente, desde a solução inicial de 25558 até 4540, depois a melhoria da solução realiza-se mais lentamente. Já depois da iteração 6000, são encontradas duas soluções, 2641 na iteração 6242 e finalmente a solução ótima.

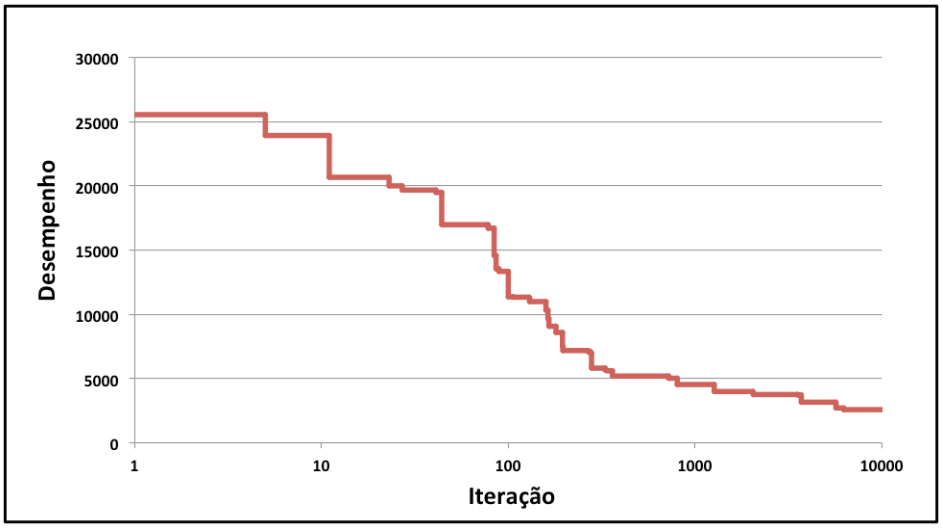

Figura 27 - Evolução do SA na Instância *wt3*

O DABC encontrou a solução ótima, 2583 em 845 iterações. Como pode ser analisado na figura 28 a solução melhora rapidamente até 2948 na iteração 364, onde são necessárias 246 iteração para encontrar uma solução com melhor desempenho. Posteriormente o DABC encontra soluções com melhor desempenho com poucas iterações de intervalo.

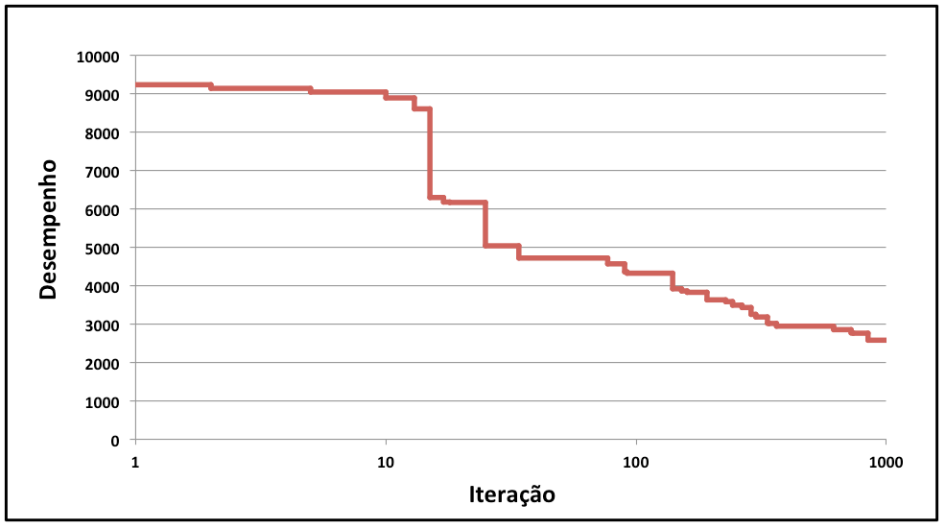

Figura 28 - Evolução do DABC na Instância *wt3*

4.7. ANÁLISE ESTATÍSTICA

Embora os resultados do estudo computacional evidenciem o melhor desempenho do DABC é necessário realizar uma análise estatística dos resultados para retirar ilações acerca do desempenho das meta-heurísticas. Para comparar o desempenho das meta-heurísticas em instâncias diferentes, os resultados serão normalizados em torno da solução ótima, isto é, serão analisadas as diferenças absolutas e proporcionais das soluções ótimas. O desvio absoluto não considera a dimensão do ótimo na normalização dos resultados, ao contrário do desvio proporcional. Isto vai permitir comparar as 30 instâncias e analisar se a diferença de desempenho entre o DABC e o SA é estatisticamente significativo.

O estudo estatístico dos resultados foi divido na análise descritiva dos resultados, que permitem analisar o desempenho das meta-heurísticas, e na inferência estatística do comportamento das meta-heurísticas em análise, que poderá permitir retirar ilações acerca do seu comportamento em problemas semelhantes. Na análise descritiva serão apresentados os parâmetros obtidos por cada meta-heurística (média, mediana e desvio padrão), que permitem obter uma indicação do desempenho nas 30 instâncias. Na inferência estatística serão utilizados testes de hipóteses para analisar se existem diferenças estatisticamente significativas no desempenho das meta-heurísticas em estudo.

4.7.1. ESTATÍSTICA DESCRITIVA

Os resultados do estudo computacional foram analisados no IBM SPSS. Os desempenhos das meta-heurísticas foram analisadas com base nos valores:

- Desempenho;
- Desvio da Solução Ótima - Absoluto;
- Desvio da Solução Ótima - Relativo.

O desvio da solução ótima, absoluto, foi calculado através da expressão 36, onde $f(x)_{MH}$ representa o desempenho da solução da meta-heurística e $f(x)_{OTM}$ o valor da solução ótima. Ao utilizar o desvio da solução ótima, absoluto, é possível comparar o desempenho em problemas distintos, não considerando o atraso da solução na comparação.

$$f(x)_{MH} - f(x)_{OTM} \tag{36}$$

Como é possível analisar na figura 29, o desempenho do SA é bastante inferior ao do DABC. O DABC encontrou soluções que estão habitualmente desviadas menos de 200 da solução ótima (a). O SA também encontrou 14 soluções desviadas menos de 200 da solução ótima (b), no entanto também encontrou um número considerável de soluções com desvios superiores, por exemplo encontrou uma solução com um desvio de 1200 (c).

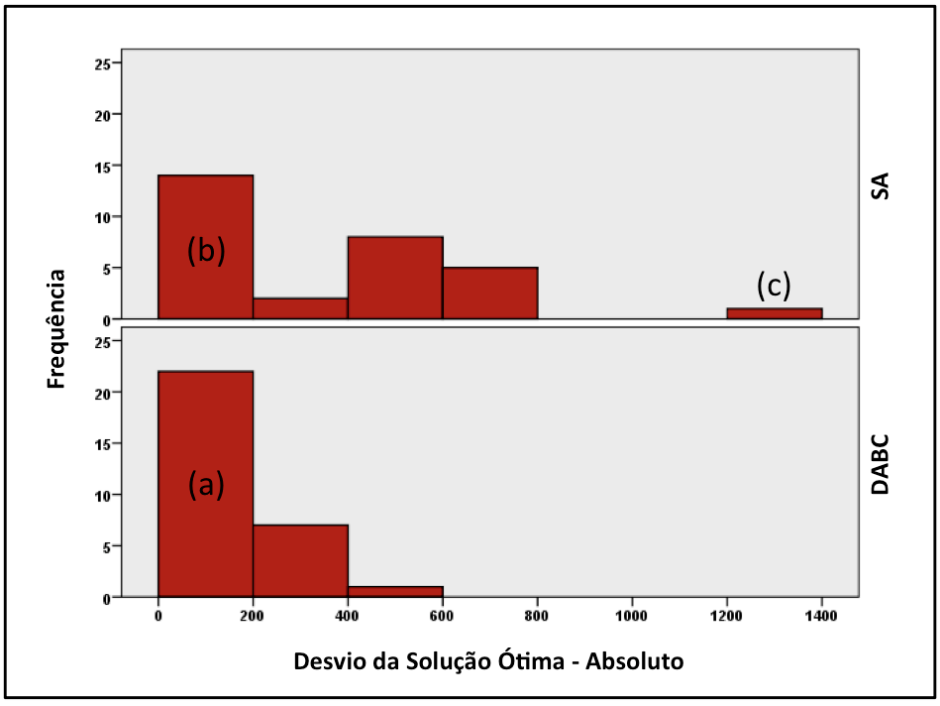

Figura 29 - Histograma do Desvio Absoluto

O SA apresenta uma mediana superior ao DABC, que é próxima do 0. O quarto quartil, no SA, é aproximadamente 750 enquanto no DABC é superior a 500. A amplitude é superior ao DABC indicando uma variação no desempenho superior no SA. É importante notar que no instância 18 o SA encontrou uma solução com um desvio do ótima muito significativa, identificado com *outlier* na figura 30. O DABC não encontra nenhum *outlier*.

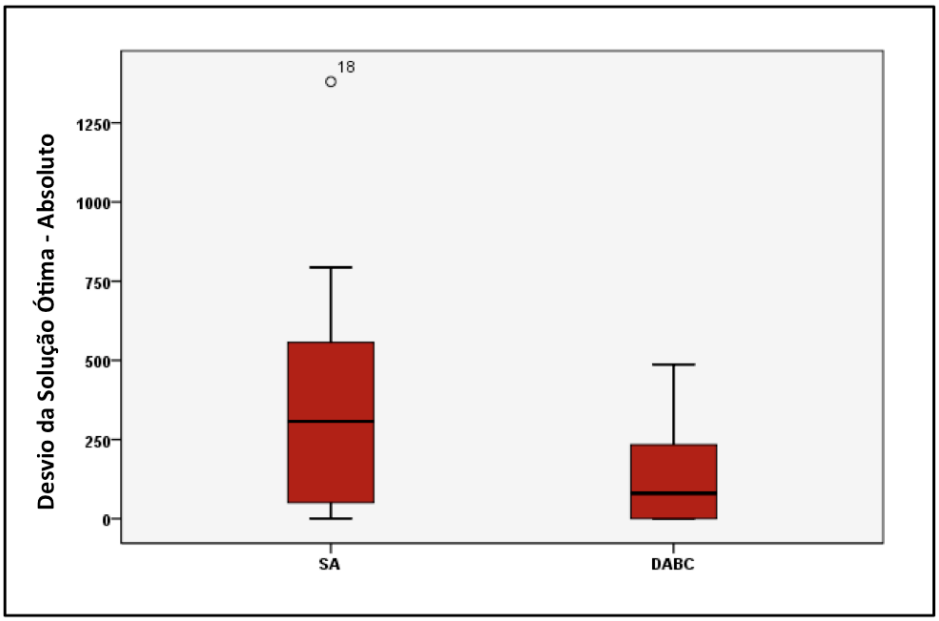

Figura 30 - *Boxplot* do Desvio Absoluto

As medidas de localização e dispersão do SA e do DABC podem ser analisados na tabela 33. O DABC obteve resultados superiores em todos os parâmetros, com um desvio médio da solução ótima de 128.87 enquanto o SA obteve um desvio médio superior, de 354.73. O DABC obteve um desvio padrão de 149.27 e uma variância de 22280.33, o SA um desvio padrão de 336.59 e uma variância de 113295.51. Os resultados demostram que além do DABC ter obtido melhores resultados também obteve menor dispersão do desempenho.

Tabela 33 - Parâmetros do Desvio Absoluto

	SA	***DABC***
Média	*354.73*	*128.87*
Mediana	*307.00*	*80.50*
Desvio Padrão	*336.59*	*149.27*
Variância	*113295.51*	*22280.33*

Analisando o *bloxplot* e a tabela 33 é possível concluir que existem evidencias estatísticas do melhor desempenho do DABC no problema em estudo, quando comparado com o SA.

O desvio da solução ótima, relativo, foi calculado através da expressão 37, onde $f(x)_{MH}$ representa o desempenho da solução da meta-heurística e $f(x)_{OTM}$ o desempenho da solução ótima. Utilizando o desvio da solução ótima, relativo, é considerado o desvio proporcional das meta-heurísticas em relação aos ótimos. Este indicador, além de permitir comparar instâncias diferentes, pondera os desvios em função da solução ótima.

$$\frac{(f(x)_{MH} - f(x)_{OTM}) \times 100}{f(x)_{OTM}} \tag{36}$$

O desempenho do DABC é sempre muito próximo do ótimo, como é possível analisar na tabela 34. O DABC encontrou soluções afastadas em menos que 1% da solução ótima em 93.3% dos problemas, e menos que 2% em 6.7% dos problemas. Os resultados do SA foram piores, em 53.3% dos problemas foram encontradas soluções afastadas em menos que 1% da solução ótima e em 13.3% dos casos foram encontradas soluções afastadas da solução ótima mais que 5%. É importante notar que as soluções do SA são muito próximas do ótimo, apenas inferiores quando comparadas com o DABC.

Tabela 34 - Frequência do Desvio Relativo

	SA			***DABC***		
	Frequência	***%***	***Cumulativa***	***Frequência***	***%***	***Cumulativa***
0% / 1%	*16*	*53.3%*	*53.3%*	*28*	*93.3%*	*93.3%*
1% / 2%	*6*	*20.0%*	*73.3%*	*2*	*6.7%*	*100%*
2% / 3%	*2*	*6.7%*	*80.0%*	-	-	-
3% / 4%	*1*	*3.3%*	*83.3%*	-	-	-
4% / 5%	*1*	*3.3%*	*86.7%*	-	-	-
5% / 6%	*3*	*10%*	*96.7%*	-	-	-
6% / 7%	*1*	*3.3%*	*100%*	-	-	-
Total	*30*	*100%*	-	*30*	*100%*	-

Utilizando o desvio relativo as diferenças do SA e DABC são mais visíveis, evidenciando o melhor desempenho do DABC, independentemente da dimensão das variáveis do problema. Considerado que soluções afastadas menos de 1% são aproximadamente ótimas, os resultados deixam entender que existe maior probabilidade de encontrar quase ótimos usando o DABC, pressuposto que será posteriormente validado, na inferência estatística.

Os resultados da tabela de frequências podem ser analisados na figura 31. O DABC obteve resultados praticamente ótimos na maioria das insâncias. O desempenho do SA é mais disperso, com soluções praticamente ótimas e soluções com desvios consideráveis.

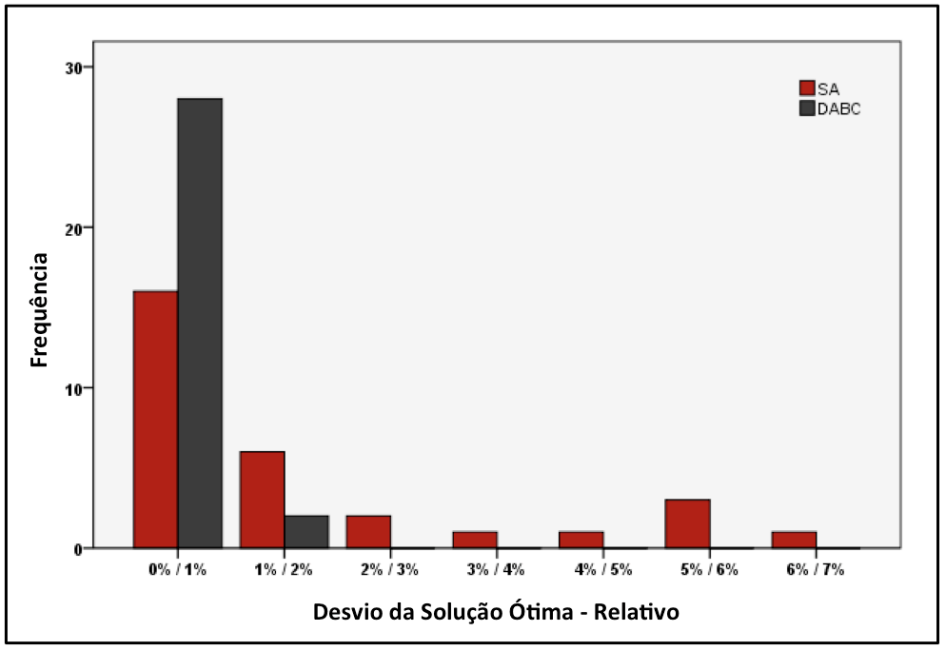

Figura 31 - Gráfico de Barras do Desvio Relativo

O SA apresenta uma amplitude muito superior ao DABC, como poderá ser analisado na figura 32. Isto é explicado pela constância nos resultados do DABC, sempre próximos do ótimo. O 4ª quartil do SA é aproximadamente 3%, e as restantes soluções *outliers*. O DABC apresenta dois *outliers*, correspondendo às soluções mais afastadas do ótimo.

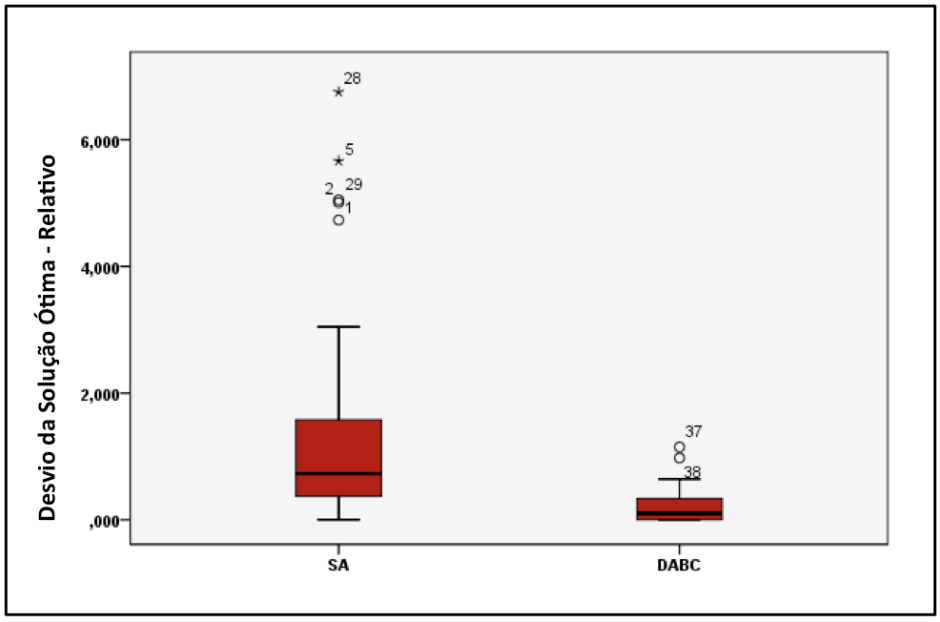

Figura 32 - *Boxplot* do Desvio Relativo

As medidas de localização e dispersão das meta-heurísticas são apresentados na tabela 35. Os resultados do SA são inferiores em todos os parâmetros, com um desvio medio 1.54% no SA e 0.22% no DABC. O SA apresenta um desvio padrão de 1.91%. e uma variância de 3.66%, enquanto o DABC apresenta um desvio padrão de 0.30% e a variância de 0.09%, isto é, têm um desempenho próximo do ótimo e praticamente constante. Da análise do *bloxplot* e dos parâmetros é possível concluir que existem evidências estatísticas do melhor desempenho do DABC no problema em estudo, quando analisado o desvio relativo.

Tabela 35 - Parâmetros do Desvio Relativo

	SA	***DABC***
Média	*1.54*	*0.22*
Mediana	*0.73*	*0.10*
Desvio Padrão	*1.91*	*0.30*
Variância	*3.66*	*0.09*

4.7.2. INFERÊNCIA ESTATÍSTICA

O SA e DABC serão comparadas com recurso ao teste de t-student, procurando inferir acerca do seu comportamento. Considerando μ_{SA} a distância média absoluta do SA da solução ótima e μ_{DABC} a distância média absoluta do DABC, as hipóteses são:

- **H_0:** $\mu_{SA} - \mu_{DABC} = 0$
- **H_1:** $\mu_{SA} - \mu_{DABC} \neq 0$

Os resultados do teste de t-student são apresentados na tabela 36. Não é possível assumir variâncias semelhantes (*p-value* de 0.000) e o teste de t-student não permite aceitar a hipótese nula (*p-value* de 0.002). É possível concluir que existem diferenças significativas entre o desempenho do SA e o DABC, quando analisado o desvio absoluto.

Tabela 36 - Teste de T-Student

	Levene's Test of Equality Variances		*t-test for Equality of Means*				
	F	*Sig.*	*t*	*df*	*Sig. (2-tailed)*	*Mean Difference*	*Std. Error Difference*
Equal variances assumed	*19.751*	*0.000*	*3.360*	*58.000*	*0.001*	*225.867*	*67.225*
Equal variances not assumed			*3.360*	*39.981*	*0.002*	*225.867*	*67.225*

O desvio médio relativo do ótimo permite ponderar o desempenho das meta-heurísticas, pela solução ótima. Isto permite, por exemplo, perceber que um desvio absoluto no 10 num problema com uma solução ótima de 1000 (1%) não é idêntico ao mesmo desvio num problema com solução ótima de 10 (100%). Considerando que soluções afastadas menos de 1% são praticamente ótimas, na tabela 37 é possível constatar que o DABC encontrou soluções quase ótimas do na maioria das instâncias, enquanto, o SA apenas encontrou soluções quase ótimas em aproximadamente 50% das instâncias.

Tabela 37 - Tabela de Contingência

	SA	***DABC***
< 1%	*16*	*28*
> 1%	*14*	*2*

Para comparar a probabilidade de encontrar soluções quase ótimas no SA e DABC será realizado o teste do qui-quadrado, cumprindo todos os pressupostos. As hipóteses são:

- **H_0:** Probabilidade de encontrar soluções praticamente ótimas é independente da meta-heurísticas utilizada
- **H_1:** Probabilidade de encontrar soluções praticamente ótimas não é independente da meta-heurísticas utilizada

Os resultados do teste, tabela 38, demonstram que a probabilidade de encontrar soluções praticamente ótimas não é independente da meta-heurística utilizada (*p-value* de 0.001), sendo possível concluir que existe maior probabilidade de encontrar soluções quase ótimas utilizando o DABC, (*p-value* de 0.001 / 2), do que recorrendo ao SA.

Tabela 38 - Teste do Qui-Quadrado

	Value	*df*	*Asymp. Sig. (2-sided))*	*Exact Sig. (2-sided)*	*Exact Sig. (1-sided)*
Pearson Chi-Square	*12.273*	*1*	*0.000*		
Continuity Correction	*10.313*	*1*	*0.001*		
Likelihood Ratio	*13.439*	*1*	*0.000*		
Fisher's Exact Test				*0.001*	*0.000*
N of Valid Cases	*60*				

4.8. CONCLUSÃO

O problema utilizado no estudo computacional, de minimização dos atrasos ponderados em máquina única, $1||\Sigma w_j T_j$, foi apresentando no início do capítulo, acompanhado pelo modelo matemático do problema. O problema procura minimizar os atrasos ponderados de um determinado número de tarefas. Foram utilizadas 30 instâncias de 50 tarefas, retiradas do *ORLibrary* [66], que também disponibiliza os respectivos resultados ótimos.

Foram selecionadas no estudo as meta-heurísticas: SA, que é uma meta-heurística de solução única muito eficiente e o DABC, que é uma adaptação recente do ABC para problemas de natureza combinatória. Foi escolhida uma implementação modular que permitisse adaptar facilmente o protótipo, para resolver outros problemas, introduzir novos mecanismos de vizinhança ou mesmo acrescentar novas meta-heurísticas.

Uma vez determinados os parâmetros, através de experiências de Taguchi, foi realizado o estudo computacional. Embora o SA e o DABC apresentem um desempenho próximo do ótimo, o DABC obteve um desempenho, quase sempre, superior ao SA, encontrou solução ótima em 11 instâncias. Em termos de tempo computacional, o SA demostrou ser mais eficiente que o DABC, tendo demorado menos de 0.001s em 24 dos 30 problemas.

Finalmente, na análise estatística foi possível comprovar o melhor desempenho do DABC, que apresenta medidas de localização e dispersão superiores, tanto no desvio da solução ótima absoluto como relativo. O teste de t-student ao não permitir aceitar a hipótese nula (*p-value* de 0.002) e demostrou existirem evidências estatísticas que o DABC têm melhor desempenho (*p-value* de 0.001 / 2). Relativamente ao desvio relativo, o teste do qui-quadrado demostrou que a probabilidade de encontrar soluções praticamente ótimas não é independente da meta-heurísticas (*p-value* de 0.001), havendo maior probabilidade de encontrar soluções quase ótimas com o DABC (*p-value* de 0.001 / 2).

5. CONCLUSÕES

O escalonamento é preponderante na sobrevivência das empresas, rentabilizando recursos produtivos e minimizando atrasos. Cumprir prazos impostos por clientes e aproveitar eficientemente o investimento em equipamentos são factores importantes para o sucesso e sobrevivência das empresas. O escalonamento distribui a produção pelos recursos e calendariza as tarefas, isto é, afecta os recursos às atividades e determina em que sequência devem ser executadas, para otimizar uma dada medida de desempenho. O escalonamento deverá ter em consideração as particularidades do sistema produtivo, por exemplo, o tipo de ambiente de fabrico, determina como é abordado o problema de escalonamento, sendo evidente que empresas com implementações distintas não distribuem a produção da mesma maneira. O escalonamento em máquina única, utilizado no estudo computacional, apenas implica decisões de sequenciação/calendarização, pois apenas existe uma máquina, isto é, um problema onde uma solução corresponde à escolha da ordem de execução das tarefas.

Embora o escalonamento possa parece simples, o número de variáveis consideradas poderá tornar a distribuição das tarefas muito complexa, devido à natureza combinatória das decisões de sequenciamento. Isto impõe um limite de tarefas, habitualmente reduzido, para o qual é possível abordar o escalonamento por técnicas enumerativas. Por exemplo no problema utilizado no estudo computacional, de minimização dos atrasos ponderados em máquina única, com 50 tarefas, existem 3.04×10^{64} ($50!$) soluções, sendo impossível enumerá-las totalmente. Para resolver problemas de escalonamento é normal recorrer-se a técnicas aproximativas, que resolvem problemas complexos, eficientemente.

Entre as técnicas de aproximação as meta-heurísticas são aquelas que têm suscitado maior interesse na teoria do escalonamento, ao fazerem uma pesquisa orientada pelas soluções. São procedimentos iterativos, que orientam heurísticas subordinadas na exploração das soluções, normalmente baseadas em fenómenos observados na natureza. Existem diversas meta-heurísticas, incluindo o *Simulated Annealing* (SA), uma meta-heurística bastante utilizada, proposta por Kirkpatrick e Cerny [33,34], inspirada no processo de tratamento térmico dos metais e na pesquisa local e o *Discrete Artificial Bee Colony* (DABC), mais recente, baseada no *Artificial Bee Colony* (ABC) proposto por Karaboga e Pham [39,40].

O estudo computacional procurou analisar e comparar o funcionamento e desempenho do SA e do DABC na resolução das instâncias do problema estudado. Para comparar as meta-heurísticas foram utilizados 30 problemas de minimização dos atrasos ponderados em máquina única, com 50 tarefas, retirados do *ORLibrary* [68]. O SA e o DABC foram posteriormente analisados, para perceber como as meta-heurística orientam a pesquisa de soluções. O desempenho das meta-heurísticas, depois de normalizado, foi analisado estatisticamente, examinando os resultados do estudo computacional e tentando inferir acerca da diferença do desempenho das meta-heurísticas em problemas semelhantes.

5.1. RESULTADOS

Os parâmetros do SA e do DABC foram determinados utilizando as experiências de Taguchi. Embora os parâmetros tenham maior impacto no desempenho quando existirem limitações temporais, durante as experiências foi possível constatar que há parâmetros que apresentam mais impacto. Tanto no SA como no DABC o mecanismo de vizinhança resultou na maior variabilidade no desempenho, os restantes parâmetros resultaram em pequenas alterações no desempenho das meta-heurísticas.

Os resultados das meta-heurísticas foram muito próximos do ótimo, particularmente tendo em consideração a dimensão do problema. O DABC encontrou 11 das 30 instâncias, 36.(6)%, enquanto o SA apenas encontrou 5 soluções ótimas, 16.(6)%. Em termos de eficiência, as meta-heurísticas concluíram a pesquisa em tempos pouco significativo. Por exemplo a instância que levou mais tempo demorou 0.032s, que é impossível de comparar com o tempo necessário para enumerar todas as soluções. Independente dos bons resultados das meta-heurísticas, o SA demora quase sempre menos tempo, sendo menor que 0.001s, impossível de medir com precisão, em 23 instâncias.

Foram analisadas as três primeiras instâncias para poder estudar as diferenças na evolução da pesquisa das meta-heurísticas. Embora ténues, com uma rápida melhoria da solução inicial que sendo aleatória deve ter várias soluções melhores na sua vizinhança, existem diferenças. O SA melhora rapidamente a solução inicial e posteriormente abrandando abruptamente a pesquisa, o DABC faz uma pesquisa mais faseada com melhorias rápidas, intervaladas pelo número de iterações. Isto é explicado pelo procedimento do DABC, que pesquisa intensamente as fonte de alimento e posteriormente espera serem abandonadas.

O resultados foram normalizados para permitir realizar uma análise de desempenho das meta-heurísticas. O desempenho superior do DABC é manifesto, encontrou mais de 20 soluções a menos de 200 da solução ótima, enquanto o SA apenas encontrou 14. Considerando o desvio ótimo relativo, o melhor desempenho do DABC é ainda mais evidente, tendo encontrado 28 soluções com desvio inferior a 1% do ótimo, enquanto o SA apenas encontrou 16. Os restantes parâmetros também demostram que o DABC apresenta um desvio ótimo reduzido. Por exemplo o desvio ótimo relativo é de 1.54% no SA e 0.22% no DABC, enquanto no desvio padrão o SA é 1.91%. e 0.03% no DABC.

No sentido de avaliar a significância dos resultados das meta-heurísticas, foram realizados dois testes de hipóteses. É importante notar que os resultados apenas permitem concluir acerca do desempenho das meta-heurísticas em instâncias com características semelhantes aos utilizados no estudo computacional. O teste de t-student não permitiu aceitar a hipótese nula (*p-value* de 0.001), demostrando que existem evidências estatísticas do desempenho superior do DABC no problema estudado. O teste do qui-quadrado, também não permitiu aceitar a hipótese nula, concluindo que o DABC têm maior probabilidade de encontrar soluções afastadas menos de 1% do ótimo (*p-value* de 0.001 / 2).

5.2. TRABALHO FUTURO

O protótipo desenvolvido, que utiliza uma implementação modular, permitirá acrescentar novas meta-heurísticas, como o TS, o ACO, o GA, ou o PSO. Isto irá permitir realizar estudos comparativos do desempenho do SA e do DABC com outras meta-heurísticas. Ao ser possível acrescentar novos mecanismos de vizinhança, que como foi demostrado podem ter muito impacto no desempenho das meta-heurísticas, tornarár-se-à possível obter resultados superiores. Finalmente o SA e DABC poderão ser comparados na resolução de outros problemas, que podem ser incorporados no protótipo desenvolvido.

Além de novos teste, com outras meta-heurísticas, mecanismos de vizinhança e problemas, uma análise interessante será a da utilização de outras técnicas de parametrização, podendo mesmo ser utilizada uma meta-heurística para parametrizar o SA e o DABC. Isto poderá levar ao desenvolvimento de um módulo de parametrização automática que, sem interação do utilizador, fará uma pesquisa pelos melhores parâmetros num problema. Tendo em consideração a morosidade da parametrização das meta-heurísticas, um módulo de parametrização automático vai tornar o protótipo muito mais eficiente.

Finalmente, pretende-se dotar o protótipo de uma ferramenta para, perante um determinado problema, escolher qual a meta-heurística a utilizar, o que é bastante pertinente quando a escolha é empírica. Utilizando uma base de dados de soluções em instâncias anteriores, o protótipo poderá ser dotado de mecanismos que recomendem uma meta-heurística que obteve um bom resultado anteriormente. Isto, aliado ao mecanismo de parametrização automático, permitirá ao utilizador introduzir os dados de um problema e deixar o protótipo propor uma meta-heurísticas e os respectivos parâmetros.

Referências Documentais

[1] Karp, R. M. (1972). *Reducibility among Combinatorial Problems*. Comexity of Computer Computations. Plenum Press.

[2] Herrman, J. H. (2006). *Handbook of Production Scheduling*. International Series in Operations Research & Management Science. Springer.

[3] Baker, K. R. & Trietsch, D. (2009). *Principles of Sequencing and Scheduling* (First Edition). Wiley.

[4] Pinedo, M. L. (2012). *Scheduling Theory, Algorithms, and Systems* (Forth Edition). Springer.

[5] Baker, K. (1974). *Introduction to Sequencing and Scheduling*. Wiley.

[6] Blazewicz, J., Ecker, H., Pesh, E., Schmidt, G. & Weglarz, J. (2007). *Handbook on Scheduling: From Theory to Applications*. International Handbooks on Information Systems. Springer.

[7] Lopez, P. & Roubellat, F. (2008). *Production Scheduling*. Control Systems, Robotics and Manufacturing. Wiley.

[8] Serra e Santos, A. (2013). *Afectação de Recursos em Sistemas de Produção*. Tese de Mestrado, Instituto Superior de Engenharia do Porto, Departamento de Engenharia Eletrotécnica.

[9] Madureira, A. M. (2003). *Aplicação de Meta-Heurísticas ao Problema de Escalonamento em Ambiente Dinâmico de Produção Discreta*. Tese de Doutoramento, Escola de Engenharia da Universidade do Minho, Departamento de Produção e Sistemas.

[10] Graham, R. L., Lawler, E. L., Lenstra, J. K. & Rinnooy Kan, A. H. G. (1979). *Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey*. Annals of Discrete Mathematics, 5, 287-326.

[11] Conway, R. W., Maxwell, W. L. & Miller, L. W. (1967). *Theory of Scheduling*. (Reprint Edition). Dover Books of Computer Science, Dover Publications.

[12] Talbi, E. G. (2009). *Meta-heuristics: From Design to Implementation*. Wiley.

[13] Santos, S. A., Varela, M. L. R., Madureira, A. M. & Ribeiro, R. A. (2014). *Parallel Machines Scheduling with Fuzzy Simulated Annealing*. Proc. of the 6th World Congress on Nature and Biologically Inspired Computing (NaBIC), 269-274.

[14] Terekhov, D., Down, D. G. & Beck, J. C. (2014). *Queueing-Theoretic Approaches for Dynamic Scheduling: A Survey*. Surveys in Operations Research and Management Science, 19(2), 105-129.

[15] Varela, M. L. R. & Ribeiro, R. A. (2014). *Distributed Manufacturing Scheduling Based on a Dynamic Multi-Criteria Decision Model*. Recent Developments in Soft Computing. Studies in Fuzziness and Soft Computing, 317, 618-623. Springer.

[16] Santos, A. S., Varela, M. L. R., Putnik, G. D. & Madureira A. M. (2014). *Alternative Approaches Analysis for Scheduling in an Extended Manufacturing Environment*. Proc. of the 6th World Congress on Nature and Biologically Inspired Computing (NaBIC), 97-102.

[17] Santos, A. S., Madureira, A. M., Varela, M. L. R., Putnik, G. D. & Abraham, A. (2015). *A Hybrid Framework for Supporting Scheduling in Extended Manufacturing Environments*. Proc. of the 14th International Conference on Hybrid Intelligent Systems (HIS), 213-218.

[18] Ibbara, O. & Kim, C. (1977). *Heuristic Algorithms for Scheduling Independent Tasks of no Identical Processors*. Journal of Association for Computing Machinery, 24(2), 280-289.

[19] Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D. & Yao, B. (2001). *A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems*. Journal of Parallel and Distributed Computing, 61(6), 810-837.

[20] Santos e Santos, A., Madureira, A. M. & Varela, M. L. R. (2015). *An Ordered Heuristic for the Allocation of Resources in Unrelated Parallel-Machines.* International Journal of Industrial Engineering Computations, 6(2), 145-156.

[21] Glover, F. (1986). *Future Paths for Integer Programing and Links to Artificial Intelligence*. Computer & Operations Research, 13(5), 533-549.

[22] Blum, C. & Roli A. (2003). *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*. ACM Computing Surveys, 35(3), 268-308.

[23] Holland, J. H. (1972). *Adaptation in Nature and Artificial Systems: An Introduction Analysis with Application to Biology, Control and Artificial Intelligence*. MIT Press.

[24] Boussaid, I., Lepagnot, J. & Siarry, P. (2013). *A Survey on Optimization Metaheuristics*. International Journal of Information Sciences, 237, 82-117.

[25] Xhafa, F. & Abraham, A. (2008). *Metaheuristics for Scheduling in Industrial and Manufacturing Applications Series: Studies in Computational Intelligence* (128). Springer.

[26] Osman, I. H. & Kelly, J. P. (1996). *Meta-Heuristics: An Overview*. Meta-Heuristics Theory and Applications. Kluwer Academic Publishers.

[27] Osman, I. H. & Laporte, G. (1996). *Metaheuristics: A Bibliography*. Annals of Operations Research, 63(5), 511-623.

[28] Dréo, J., Petrowski, A., Siarry, P. & Taillard, E. (2006). *Metaheuristics for Hard Optimization: Methods and Case Studies*. Springer.

[29] Pirlot, M. (1996). *General Local Search Methods*. European Journal of Operational Research, 92, 493-511.

[30] Orlin, J. B., Abraham, P. P. & Shulz, A. S. (2004). *Approximated Local Search in Combinatorial Optimization*. Proc of the 15th annual ACM-SIAM symposium on Discrete Algorithms (SODA), 587-596.

[31] Madureira, A., Pereira, I. & Sousa, N. (2010). *Collective Intelligence on Dynamic Manufacturing Scheduling Optimization*. Proc. of the IEEE 5th International Conference on Bio-Inspired Computing: Theories and Applications, 1693-1697.

[32] Metropolis, W., Roenbluth, A. & Roenbluth, M. (1953). *Equations of the State Calculations by Fast Computing Machines*. Journal of Chemical Physics, 21, 1087-1092.

[33] Kirkpatrick, S., Gelatt, C. D. & Vecchi, P. M. (1983). *Optimization by Simulated Annealing*. Science, 220, 671-680.

[34] Cerny, V. (1985). *A Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulated Annealing Algorithm*. Journal of Optimization Theory and Applications, 45, 41-51.

[35] Madureira, A. M. (2010). *Técnicas Emergentes de Optimização no Suporte à Tomada de Decisão*. Texto Lição para provas publicas de Professor Coordenador, Instituto Superior de Engenharia do Porto, nº 694/2009 publicado na II Série do D.R. nº 138 de 20 de Julho de 2009.

[36] Glover, F. (1986). *Future Paths for Integer Programming and Links to Artificial Intelligence*. Computers and Operations Research, 1, 533-549.

[37] Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano.

[38] Stutzle, T., López-Ibáñez, M., Pellegrini, P., Maur, M., Montes de Oca, M., Birattari, M., & Dorigo, M. (2010). *Parameter Adaptation in Ant Colony Optimization*. Technical Report No. TR/IRIDIA/2010-002, Université Libre de Bruxelles.

[39] Karaboga, D. (2005). *An Idea Based on Honey Bee Swarm for Numerical Optimization*. Technical Report-TR06, Erciyes University, Engineering Faculty.

[40] Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S. & Zaidi, M. (2005). *The Bees Algorithm*. Technical Note, Cardiff University.

[41] Karaboga, D. & Gorkemli, B. (2011). *A Combinatorial Artificial Bee Colony Algorithm for Traveling Salesman Problem*. Proc. of the International Symposium on Innovations in Intelligent Systems and Applications (INISTA), 50-53.

[42] Pan, Q., Tasgetiren, M. F., Suganthan, P. N. & Chua, T. J. (2011). *A Discrete Artificial Bee Colony Algorithm for the Lot-Streaming Flow Shop Scheduling Problem*. Information Sciences, 181, 2455-2468.

[43] Li, X. & Yin, M. (2012) *A Discrete Artificial Bee Colony Algorithm with Composite Mutation Strategies for Permutation Flow Shop Scheduling Problem*. Scientia Iranica, 19(6), 1921-1935.

[44] Liu, Y. F. & Liu, S. Y. (2013) *A Hybrid Discrete Artificial Bee Colony Algorithm for Permutation Flowshop Scheduling Problem*. Applied Soft Computing, 13, 1459-1463.

[45] Holland, J. H. (1972). *Adaptations in Nature and Artificial Systems: An Introductory Analysis with Application to Biology, Control and Artificial Intelligence*. MIT Press.

[46] Kennedy, J. & Eberhart, R. (1995). *Particle Swarm Optimization*. Proc. Of the IEEE International Conference on Neural Networks, 1942-1948.

[47] Hutter, F., Hoos, H., Leyton-Brown, K. & Murphy, K. P. (2009). *An Experimental Investigation of Model-Based Parameters Optimisation: SPO and Beyond*. Proc. of the 11th Annual Conference on Genetic and Evolutionary Computation, 271-278.

[48] Adenso-Diaz, B. & Laguna, M. (2006). *Fine-Tuning of Algorithms using Fractional Experiment Design and Local Search*. Operations Research, 54(1), 99-114.

[49] Montero, E., Riff, M. & Neveu, B. (2014). *A Beginner's Guide to Tuning Methods*. Applied Soft Computing, 17, 39-51.

[50] Birattari, M., Stutzle, T., Paquete, L. & Varrentrapp, K. (2002). *A Racing Algorithm for Configuring Metaheuristics*. Proc. of the Conference on Genetic and Evolutionary Computation, 11-18.

[51] Nannen, V. & Eiben, A. E. (2007). *Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters*. Proc. of the International Joint Conference for Artificial Intelligence, 975-980.

[52] Hutter, F., Hoos, H. & Stutzle, T. (2007). *Automatic Algorithm Configuration Based on Local Search*. Proc. of the Conference on Artificial Intelligence, 1152-1157.

[53] Bartz-Beielstein, T., Lasarczyk, C. W. G. & Preuss, M. (2005). *Sequential Parameter Optimization*. Proc of the IEEE Congress on Evolutionary Computing, 773-780.

[54] Nadir, B., Zandieh, M. & Fatemi Ghomi, S. M. T. (2009). *Scheduling Job Shop Problems with Sequence-Dependent Setup Times*. International Journal of Production Research, 47(21), 5959-5976.

[55] Zandieh, M., Amiri, M., Vahdani, B. & Soltani, R. (2009). *A Robust Parameter Design for Multi-Response Problems*. Journal of Computational and Applied Mathematics, 230, 463-476.

[56] Markovic, D., Petrovic, G., Cojbasic, Z. & Marinkavic D. (2012). *A Comparative Analysis of Metaheuristic Maintenance Optimization of Refuse Collection Vehicles Using the Taguchi Experimental Design*. Transaction of FAMENA, 4(36), 25-38.

[57] Maleki-Daronkolaei, A. & Seyedi, I. (2013). *Taguchi Method for Three-Stage Assembly Flow Shop Scheduling Problem with Blocking and Sequence-Dependent Set Up Times*. Journal of Engineering Science and Technology, 8(5), 603-622.

[58] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. & Qu, R. (2013). *Hyper-Heuristics: A Survey of the State of the Art*. Journal of the Operations Research Society, 64, 1695-1724.

[59] Pereira, I., Madureira, A. & Olivera, P. (2013). *Meta-Heuristics Self-Parameterization in a Multi-Agent Scheduling System Using Case-Based Reasoning.* Computational Intelligence and Decision Making, 61, 99-109.

[60] Pereira, I., Madureira, A., Olivera, P. & Abraham, A. (2013). *Tuning Meta-Heuristics Using Multi-Agent Learning in a Scheduling System.* Transactions on Computational Science, 190-210. Springer.

[61] Chakhlevitch, K. & Cowling, P. (2008). Hyperheuristics: Recent Developments. Adaptive and Multilevel Metaheuristics, Studies in Computational Intelligence, 136, 3-29.

[62] Madureira, A., Cunha, B. & Pereira, I. (2014). *Cooperation Mechanism for Distributed Resources Scheduling for Artificial Bee Colony based Self-Organized Scheduling Systems*. Proc. of the IEEE Congress on Evolutionary Computation, 565-572.

[63] Madureira, A., Pereira, I. & Falcão, D. (2013). *Cooperative Scheduling System with Emergent Swarm Based Behavior*. Advanced in Information Systems and Technologies, 661-671.

[64] Burke, E. K., Kendall, G. & Soubeiga, E. (2003). *A Tabu-Search Hyperheuristic for Timetabling and Rostering*. Journal of Heuristics, 9(6), 451-470.

[65] Vázquez-Rodríguez, J., Petrovic, S. & Salhi, A. (2007). *A Combined Meta-Heuristic with Hyper-Heuristic approach to the Scheduling of the Hybrid Flow Shop with Sequence Dependent Setup Times and Uniform Machines*. Proc. of the International Conference on Scheduling: Theory & Applications, 506-513.

[66] Quelhadj, D. & Petrovic, S. (2010). *A Cooperative Hyper-Heuristic Search Framework*. Journal of Heuristics, 16, 835-857.

[67] Baker, K. R. & Keller, B. (2010). *Solving the Single-Machine Sequencing Problem Using Integer Programming*. Computer & Industrial Engineering, 59(4), 730-735.

[68] Keha, A. B., Khowala, K. & Fowler, J. W. (2009). *Mixed Integer Programming Formulations for Single Machine Scheduling Problems*. Computers & Industrial Engineering, 56, 357-367.

[69] Khowala, K., Keha, A. & Fowler, J. (2005) *A Comparison of Different Formulations for the Non-Preemptive Tardiness Scheduling Problem*. Proc. International Conference on Scheduling: Theory & Applications, 643-651.

[70] Beasley, J. E. (1990). *ORLibrary*: http://www.brunel.ac.uk/~mastjjb/jeb/info.html.

[71] Park, M. W. & Kim, Y. D. (1998). *A Systematic Procedure for Setting Parameters in Simulated Annealing Algorithms*. Computers & Operations Research, 25(3), 207-217.

[72] Eglese, R. W. (1990). *Simulated Annealing: A Tool for Operational Research*. European Journal of Operational Research, 46, 271-281.

[73] Rose, J., Klebsch, W. & Wolf, J. (1990). *Temperature Measurement and Equilibrium Dynamics of Simulated Annealing Placement*. IEEE Transactions on Computer Aided Design, 9, 253-259.

[74] Chek, K. M., Goldberg, J. B. & Askin, G. (1991). *A Note on the Effect Neighborhood Structure in Simulated Annealing*. Computers & Operations Research, 18, 537-547.

[75] Yan, G. & Li, C. (2011). *An Effective Refinement Artificial Bee Colony Optimization Algorithm Based on Chaotic Search and Application for PID Control Tuning*. Journal of Computational Information Systems, 7(9), 3309-3316.

[76] Akay, B. & Karaboga, D. (2009). *Parameter Tuning for the Artificial Bee Colony Algorithm*. Computational Collective Intelligence. Semantic Web, Social Network and Multiagent Systems. Lecture Notes in Computer Science, 5796, 608-619.

[77] Kockanat, S. & Karaboga, N. (2013). *Parameter Tuning of Artificial Bee Colony Algorithm for Gaussian Noise Elimination on Digital Images*. Proc. of the International Symposium on Innovation in Intelligent Systems and Applications (INISTA), 1-4.

[78] Kiran, M. S. & Gunduz, M. (2014). *The Analysis of Peculiar Control Parameters of Artificial Bee Colony Algorithm on the Numerical Optimization Problems*. Journal of Computer and Communications, 2, 127- 136.