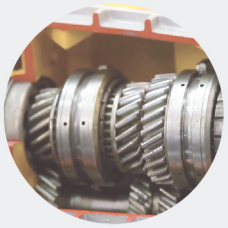
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA E GESTÃO INDUSTRIAL






PROGRAMAÇÃO DE MÁQUINAS PARALELAS DEDICADAS COM FAMÍLIAS DE SETUPS, RECURSOS ADICIONAIS E DATAS DE ENTREGA

ANA RITA BARBOSA MOURA MOTA FERREIRA
julho de 2023

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

PROGRAMAÇÃO DE MÁQUINAS PARALELAS DEDICADAS COM SETUPS DEPENDENTES DA SEQUÊNCIA DE FAMÍLIAS, RECURSOS ADICIONAIS E DATAS DE ENTREGA

Ana Rita Barbosa Moura Mota Ferreira

2023

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Mecânica

P.PORTO

PROGRAMAÇÃO DE MÁQUINAS PARALELAS DEDICADAS COM SETUPS DEPENDENTES DA SEQUÊNCIA DE FAMÍLIAS, RECURSOS ADICIONAIS E DATAS DE ENTREGA

Ana Rita Barbosa Moura Mota Ferreira

Estudante n.º 1180733

Dissertação apresentada ao Instituto Superior de Engenharia do Porto para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia e Gestão Industrial, realizada sob a orientação do Professor Doutor Manuel Joaquim Pereira Lopes.

2023

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Mecânica


isep
Instituto Superior de
Engenharia do Porto


P.PORTO


isep

AGRADECIMENTOS

A execução deste projeto apenas foi possível devido ao contributo direto ou indireto de algumas pessoas a quem direciono as seguintes palavras de gratidão.

Ao Professor Doutor Manuel Pereira Lopes, por todo o apoio dado ao longo da execução do mesmo, pela paciência e transmissão de conhecimentos sempre que necessário para o progresso do projeto, permanecendo sempre disponível, desafiando-me constantemente e sendo, por isso, peça fundamental para o resultando atingido.

Aos meus pais por estarem sempre presentes ao longo de toda esta jornada, colocando-me sempre em primeiro lugar, apostando no meu futuro, sendo flexíveis e dando-me motivação para perseguir os meus sonhos.

Ao meu namorado Ricardo por ser um pilar de estabilidade, acreditar sempre em mim e me ajudar a superar as fases mais complicadas.

Por fim, aos meus amigos e restantes familiares pela amizade, ajuda e carinho, tornando esta meta menos difícil de alcançar

página propositadamente em branco

RESUMO

A empresa INPLAS procura melhorar o planeamento de produção nas suas fábricas. Nesse sentido, o principal objetivo deste trabalho foi o desenvolvimento de um sistema de apoio à decisão para resolver o desafio específico de programação de máquinas paralelas dedicadas, considerando *setups* dependentes da sequência de famílias, recursos adicionais e datas de entrega (PMSR).

Primeiro foram estudados e adaptados dois modelos matemáticos: *Strip-Packing* e *Time Index*. As adaptações referidas prendem-se com considerações necessárias para responder à realidade do problema, nomeadamente, conversão dos modelos gerais para o caso particular de máquinas dedicadas, introdução da configuração inicial das máquinas, inclusão das datas de entrega e a capacidade de lidar com *setups* dependentes da sequência de famílias. Além disso propõe-se uma função multi-objetivo para a minimização do *tardiness* e da soma dos *makespan* de todas as máquinas. A programação do modelo funciona de forma lexicográfica pelo método de duas fases, resolvendo na 1ª fase o modelo com vista à minimização do *tardiness* e na 2ª fase objetivando a minimização da soma dos *makespan* de todas as máquinas. Após constatar que o modelo *Strip-Packing* apresentou um desempenho superior em relação ao modelo *Time Index*, a análise concentrou-se exclusivamente no *Strip-Packing*.

Neste trabalho são também desenvolvidas heurísticas matemáticas para atingir melhores resultados visto que o modelo matemático só se mostrou capaz de resolver instâncias de pequena dimensão. Dentro deste paradigma, as estratégias utilizadas foram o *warm-start* (fornecimento de uma solução inicial válida que atua como *upper bound*) e um *lower bound* (limita o espaço soluções inferiormente). Assim, foram desenvolvidas uma heurística inicial para limitar inferiormente a soma dos *makespan* de todas as máquinas *e* uma metaheurística GRASP para gerar soluções iniciais válidas.

As heurísticas matemáticas foram eficazes, com destaque para a combinação da heurística construtiva e da metaheurística GRASP com modelo matemático. Para as instâncias pequenas, a 1ª fase, que procura a minimização do *tardiness*, permitiu encontrar a solução ótima em 75% dos testes executados, tendo um gap médio de 4,40% e um ganho de 31,57% em gap e 24,82% em tempo computacional face ao comportamento isolado do modelo. A 2ª fase, que procura a minimização da soma dos *makespan* de todas as máquinas também mostrou resultados positivos, atingindo um gap médio de 3,89% e ganhos de 84,87% em gap e 9,66% em tempo computacional face ao modelo isolado, encontrando a solução ótima em 37,5% dos testes executados com as instâncias pequenas.

Os resultados obtidos pelo GRASP evidenciam a qualidade das suas soluções. Em 37,5% das instâncias, a solução do GRASP foi a solução ótima de *tardiness*, sendo 4,4% o desvio médio da solução GRASP à solução ótima. A heurística associada ao *lower bound* superou a relaxação linear, valorizando a sua introdução. Relativamente aos limites computacionais, o GRASP apresenta bons resultados para instâncias pequenas, no entanto, nas maiores instâncias pode atingir tempos computacionais até 50 minutos, valores consideravelmente elevados. Por outro lado, a heurística construtiva para *lowerbound* produz soluções em menos de 1 segundo para qualquer dimensão.

PALAVRAS-CHAVE

Máquinas Paralelas, Datas de Entrega, Recursos, *Setups*, Heurísticas Matemáticas, GRASP.

Página propositadamente em branco

ABSTRACT

The company INPLAS aims to improve production planning in its factories. In this regard, the main objective of this work was to develop a decision support system to address the specific challenge of scheduling dedicated parallel machines, considering sequence-dependent family setups, additional resources, and delivery dates (PMSR).

Initially, two mathematical models, Strip-Packing and Time Index, were studied and adapted. The adaptations made were necessary to address the problem's reality, including converting the general models to the specific case of dedicated machines, introducing the initial machine configuration, incorporating delivery dates, and handling sequence-dependent family setups. Additionally, a multi-objective function was proposed to minimize tardiness and the sum of makespans for all machines. The model was programmed using a lexicographic approach with a two-phase method, solving the first phase for minimizing tardiness and the second phase for minimizing the sum of makespans for all machines. After determining that the Strip-Packing model outperformed the Time Index model, the analysis focused exclusively on Strip-Packing.

In this work, mathematical heuristics were also developed to achieve better results, as the mathematical model alone proved capable of solving only small-sized instances. Within this paradigm, the strategies employed were warm-start (providing an initial valid solution that acts as an upper bound) and a lower bound (limiting the solution space from below). Thus, an initial heuristic was developed to lower bound the sum of makespans for all machines, and a GRASP metaheuristic was implemented to generate valid initial solutions.

The mathematical heuristics proved effective, particularly the combination of the constructive heuristic and the GRASP metaheuristic with the mathematical model. For small instances, the first phase, which aimed to minimize tardiness, found the optimal solution in 75% of the tests performed, with an average gap of 4.40% and a gap reduction of 31.57% and computational time improvement of 24.82% compared to the isolated model. The second phase, which aimed to minimize the sum of makespans for all machines, also yielded positive results, with an average gap of 3.89% and gap reductions of 84.87% and 9.66% in computational time compared to the isolated model, finding the optimal solution in 37.5% of the tests performed with small instances.

The results obtained by GRASP demonstrate the quality of its solutions. In 37.5% of the instances, the GRASP solution was the optimal tardiness solution, with an average deviation of 4.4% from the optimal solution. The heuristic associated with the lower bound outperformed the linear relaxation, highlighting its effectiveness. Regarding computational limits, GRASP showed good results for small instances; however, for larger instances, it can reach computational times of up to 50 minutes, which are considerably high values. On the other hand, the constructive heuristic for the lower bound produces solutions in less than 1 second for any dimension.

KEYWORDS

Parallel machines, Due Dates, Resources, Setups, Matheuristics, GRASP

página propositadamente em branco

ÍNDICE

página propositadamente em branco

ÍNDICE DE FIGURAS

página propositadamente em branco

ÍNDICE DE TABELAS

página propositadamente em branco

LISTAS DE SIGLAS

CP	*Constraint Programming*
CPLOT	*Center for Production and Logistics Optimization Technologies*
GRASP	*Greedy Randomized Adaptive*
ISEP	Instituto Superior de Engenharia do Porto
MILP	*Mixed Integer Linear Programming*
PMFRS	*Parallel Machines with Flexible Resources Scheduling problem*
PM	*Parallel Machines schedulling problem*
PMR	*Parallel Machines with Resources scheduling problem*
PMS	*Parallel Machines with Setups scheduling problem*
PMSR	*Parallel Machines with Setups and Resources scheduling problem*
P.Porto	Instituto Politécnico do Porto
UPMFRS	*Unspecified Parallel Machines with Flexible Resource Scheduling problem*

página propositadamente em branco

1. INTRODUÇÃO

Ao longo deste capítulo será descrito o trabalho proposto, contextualizando-se devidamente o projeto. Deste modo, o capítulo encontra-se subdividido em três partes: Problema de investigação, enquadramento e pertinência, Questão e objetivos da investigação e Opções metodológicas. No primeiro subcapítulo explora-se o problema, enquadrando brevemente a empresa e os principais aspetos da literatura. Seguidamente, é enunciada a questão de investigação bem como os objetivos principais e específicos que guiarão a execução do projeto. Por fim, traça-se a metodologia e a estratégia de investigação a seguir.

1.1. Problema de investigação, enquadramento e pertinência

O Grupo Simoldes é um grupo empresarial português, localizado em Oliveira de Azeméis, que se dedica à fabricação de moldes de injeção para a indústria dos plásticos (*Simoldes Plastics*, n.d.). A INPLAS é uma das empresas que integra este grupo e tem como sua principal atividade a injeção de termoplásticos que serão maioritariamente utilizados na indústria automóvel (Inplas – Simoldes, n.d.).

O Grupo Simoldes mostra-se interessado pela pesquisa e inovação, engenharia e desenvolvimento, com o objetivo de terem a capacidade tecnológica para poderem continuar a oferecer produtos de elevada qualidade e que correspondem às necessidades e tendências dos clientes. Esta política também permite que se mantenham competitivos e sustentáveis (*Simoldes Plastics*, n.d.). Nesse sentido, este projeto surge como um desafio da INPLAS proposto ao *Center for Production and Logistics Optimization Technologies* (CPLOT) com o objetivo de otimizar o sistema de produção, permitindo que o planeamento das máquinas paralelas seja suportando por um sistema de apoio à decisão capaz de gerar diferentes cenários de forma rápida.

Assim, a INPLAS tem por objetivo responder à procura diária de peças, considerando as restrições de capacidade e elegibilidade de máquina bem como disponibilidade do operador. Na fábrica, existem três tipos de recursos adicionais além das máquinas: moldes, operadores de máquinas e equipas de *setup*. Cada molde apenas poderá ser utilizado numa só máquina, mas poderá produzir diferentes produtos. Peças que partilham o mesmo molde são consideradas da mesma família e não existirá *setup* entre elas.

Os operadores de máquinas são necessários durante o processamento das tarefas e as equipas de *setup* durante a operação de troca de molde. Além dos aspetos habituais dos problemas estudados de máquinas paralelas, com o objetivo de uma maior aproximação da realidade vivida na indústria, será incorporado o estado inicial da máquina. O estado inicial de uma máquina em cada dia permite reconhecer: o molde presente na máquina no início do dia e a duração da tarefa em produção do dia anterior (caso se aplique). Datas de entrega foram também introduzidas no problema como um fator a considerar no processo de escalonamento, objetivando-se o menor atraso possível.

O problema descrito enquadra-se como sendo um problema de escalonamento de máquinas paralelas com famílias de *setup*s dependentes da sequência, recursos (de *setup* e de processamento) e datas de entrega. Os problemas gerais são muito complexos pois integram três subproblemas: alocação – atribuição das tarefas às máquinas onde serão executadas, sequenciação – definição da ordem de execução das tarefas em cada máquina, e tempo – período exato em que

determinada tarefa será processada. Quando adaptado a máquinas paralelas dedicadas, o problema não requere tomada de decisão ao nível da alocação.

Este problema retrata um contexto real, complexo e muito difícil de resolver, tornando a sua resolução uma vantagem competitiva para a empresa. Saliente-se que a complexidade referida dificulta a obtenção de soluções ótimas com tempos computacionais razoáveis. Por este motivo, utilizar-se-ão heurísticas matemáticas (*matheuristics)* – uma combinação de heurísticas e modelos matemáticos.

A pertinência deste projeto relaciona-se com a necessidade de tornar as fábricas mais sustentáveis, eficazes e eficientes, sendo crucial o desenvolvimento de ferramentas inteligentes baseadas em técnicas de otimização que apoiem o processo de tomada de decisão.

1.2. Questão e objetivos de investigação

Tendo em conta a descrição do problema anteriormente mencionado, o presente trabalho tem como finalidade responder às seguintes questões: Para o caso particular de máquinas paralelas dedicadas, será um modelo indexado ao tempo capaz de superar um modelo *Strip-Packing*? Qual será o impacto das heurísticas matemáticas na eficiência do algoritmo utilizado?

Em resposta à questão de investigação, define-se o seguinte objetivo geral: desenvolvimento de um sistema de apoio à decisão para o planeamento de máquinas paralelas com limite de recursos e *setups* dependentes da sequência.

Para a prossecução do objetivo geral desta investigação, formularam-se os seguintes objetivos específicos:

- Adaptação e extensão do modelo *Strip-Packing* e *Time Index* de (Fanjul-Peyro, 2020) para o caso particular das máquinas paralelas dedicadas com *setups* dependentes da sequência de famílias, recursos adicionais e datas de entrega;
- Desenvolvimento de heurísticas para obter soluções iniciais;
- Implementação do modelo (a partir da biblioteca *Pulp*) e das heurísticas em linguagem *Pyhton*;
- Estudar o desempenho da heurística matemática.

1.3. Opções metodológicas

No seguimento dos objetivos traçados, opta-se pelo enquadramento sugerido por (Coutinho, 2011) para a organização das opções metodológicas, isto é, estabelece-se uma relação hierárquica entre técnicas, métodos e metodologias. Assim, num primeiro nível estão as técnicas específicas (a aplicação prática) e no segundo nível o método, onde se agrega uma panóplia de técnicas comuns a uma determinada área científica. No último nível, o mais geral, "a metodologia descreve e analisa os métodos, distancia-se da prática para poder tecer considerações teóricas em torno do seu potencial na produção de conhecimento científico" (Coutinho, 2011).

É consensual para (Coutinho, 2011) e (Stockemer, 2019) que a maioria dos autores distingue duas grandes perspetivas metodológicas: quantitativa e qualitativa. A perspetiva quantitativa centra-se na análise de fenómenos bem como na medição/avaliação de variáveis, com o intuito de os prever, perceber e controlar. Por outro lado, a perspetiva qualitativa deve estudar intenções e situações,

através do método indutivo, sem que se imponha referenciais teóricos. Neste sentido, o presente trabalho alinha-se com a perspetiva quantitativa visto que o objetivo passa pela realização de testes que possam comprovar a hipótese colocada (Coutinho, 2011).

Relativamente à estratégia de investigação, (Fernandes et al., 2020) menciona várias alternativas, tais como: investigação experimental, inquérito, estudo de caso, investigação-ação. O presente trabalho segue a estratégia investigação-ação uma vez que o objetivo é a compreensão de problemas práticos e reais na organização e a aplicação de mudanças. Esta estratégia permite partir dos fundamentos teóricos, aplicando-os e validando (ou refutando) a sua pertinência em determinado contexto, procurando que, idealmente, se consiga melhorar uma situação real encontrada (Coutinho, 2011; Fernandes et al., 2020).

1.4. Estrutura do trabalho

Além da Introdução, capítulos como a Revisão Bibliográfica, Métodos e Aplicação, Resultados e Discussão e Conclusão compõe este trabalho.

A Revisão Bibliográfica fornece uma revisão da literatura para contextualizar o projeto, permitindo a sua classificação como um problema de máquinas paralelas dedicadas com *setups* dependentes da sequência de famílias e recursos adicionais. Ao longo do capítulo aborda-se a importância de incorporar configurações e sua dependência de sequência, bem como o papel e os tipos de recursos. Neste capítulo também se explora as heurísticas matemáticas, enfatizando as estratégias de *warm-start*.

Em Métodos e Aplicação é apresentado a descrição do problema com detalhe, as principais contribuições, dois modelos matemáticos (*Strip-Packing* e *Time Index*) e diferentes abordagens de heurísticas matemáticas, nomeadamente técnicas de *warm-start* e *lower bound*.

No capítulo Resultados Computacionais apresentam-se resultados, comparam-se modelos (*Strip-Packing* vs. *Time Index*), analisa-se resultados computacionais e avalia-se o desempenho da heurística e metaheurística implementadas.

Na Conclusão são destacados os principais contributos e conclusões do estudo, discute-se perspetivas futuras e abordam-se os desafios enfrentados.

Complementam estes capítulos o Resumo, os Índices e Lista de Siglas, as Referências Bibliográfica, os Apêndices e os Anexos.

2. REVISÃO BIBLIOGRÁFICA

Este capítulo é dedicado à contextualização do projeto através de um processo de revisão da literatura. O subcapítulo inicial tem como objetivo a realização de um enquadramento e classificação das máquinas paralelas. Posteriormente é dedicado um subcapítulo aos *setups*, justificando a importância de os incorporar e o modo como podem ser afetados pela sequência. No subcapítulo 2.3. reflete-se sobre o papel dos recursos e as suas tipologias. No quarto subcapítulo são somados os conceitos de máquinas paralelas, recursos adicionais e *setups* dependentes da sequência. Heurísticas Matemáticas são o mote do subcapítulo cinco, dando-se principal destaque à estratégia de *warm-start*, identificando-se as vantagens da utilização da abordagem. Os subcapítulos 2.6 e 2.7 destinam-se, respetivamente, ao estudo dos indicadores de desempenho e *solvers,* permitindo identificar várias alternativas, conhecendo as tendências atuais. Por fim, o último subcapítulo apresenta com detalhe os modelos matemáticos de referência para este estudo.

2.1. Problema de Escalonamento de Máquinas Paralelas

"O escalonamento de operações é uma das questões mais críticas no planeamento e gestão dos processos de fabrico" (Pezzella et al., 2008). Um dos problemas mais estudados no âmbito do escalonamento de operações é o problema de escalonamento de máquinas paralelas, sendo que este se debruça na necessidade de atribuir tarefas às máquinas disponíveis, percebendo quando é que estas serão, de facto, processadas.

O problema de máquinas paralelas define-se como um conjunto de n tarefas que serão executadas num outro conjunto de m máquinas. Neste problema torna-se relevante proceder à categorização do conjunto de máquinas uma vez que esta categorização pode permitir particularizar o problema. Assim, segundo (Kellerer & Strusevich, 2003; Vakhania et al., 2014), existem as seguintes tipologias de classificação de máquinas:

- Idênticas – o tempo de processamento de uma tarefa é igual em cada uma das máquinas;
- Uniforme – cada máquina tem a sua própria velocidade, sendo possível identificar o tempo necessário para processar uma determinada tarefa em qualquer máquina, desde que se conheça o tempo de processamento dessa mesma tarefa para uma máquina: $p_{ij} = {p_{1j}}/{s_i}$;
- Não relacionadas – a velocidade da execução de cada tarefa varia conforme a máquina, sendo necessário conhecer o tempo de processamento de cada tarefa em cada máquina (caso geral);
- Dedicadas – à priori, já se conhece o subconjunto de tarefas que serão processadas em cada uma das máquinas (extingue-se o subproblema de alocação).

Outro aspeto a afetar a atribuição das tarefas às máquinas são as condições de elegibilidade. Segundo (Pinedo, 1995), estas condições são essenciais quando uma tarefa não pode ser executada em qualquer uma das máquinas disponíveis, mas apenas num subconjunto de máquinas.

Considerando que a elegibilidade traduz situações em que as tarefas apenas podem ser executadas num subconjunto de máquinas, considera-se que o caso das máquinas dedicadas seja um caso particular desta elegibilidade uma vez que aqui também as tarefas apenas podem apenas ser executadas por um subconjunto de apenas 1 máquina.

2.2. *Setups*

Ao tempo requerido para a preparação dos recursos necessários à execução de uma tarefa (máquina, pessoas, ferramentas, etc) comummente atribuímos o nome de tempo de *setup* (Allahverdi, 2015). Os tempos e recursos de *setup* podem estar associados a diferentes tipos de ações: ajustes, limpezas, reconfigurações, entre outros. Apesar da sua importância, as atividades de *setup* não atribuem qualquer valor agregado aos produtos, tornando ainda mais importante a busca pela melhor utilização deste tipo de recursos (Allahverdi, 2015).

Em (Allahverdi & Soroush, 2008) são identificadas cerca de 50 diferentes indústrias nas quais é pertinente o cuidado com processos de *setup*, o que nos permite compreender que será um fator transversal às demais indústrias. Apesar destes dados, a pesquisa de (Allahverdi, 2015) revelou que em mais de 90% dos artigos publicados os *setups* não são considerados.

Acrescente-se ainda que os *setups* podem ser classificados como independentes ou dependentes da sequência (Allahverdi et al., 1999):

- *Setup* independente da sequência: o custo/tempo associado depende apenas da própria tarefa e, neste caso, não há motivos para separar o tempo de *setup* do tempo de processamento, sendo possível somá-los previamente;
- *Setup* dependente da sequência: o custo/tempo não depende apenas da própria tarefa mas da sequência de tarefas e o mais correto será tratar o tempo de *setup* e o tempo de processamento como parâmetros isolados.

Apesar da existência das duas tipologias, na indústria é mais comum os tempos de *setup* variarem com a sequência. Um exemplo prático é identificado na indústria dos plásticos: aquando da necessidade de mudança de cor é necessário algum tempo até que o material extraído tenha a cor pretendida e, noutras fábricas onde são utilizadas tintas, a meticulosidade associada à limpeza pode variar conforme a tinta subsequente e a sucessora. Este e outros exemplos que reiteram o impacto da sequência nos tempos de *setup* são possíveis de encontrar em (Tavakkoli-Moghaddam et al., 2009).

2.2.1. Programação de Máquinas Paralelas com *Setups* Dependentes da Sequência

O PMS (problema do escalonamento de Máquinas Paralelas com *Setups* dependentes da sequência) surge como uma versão mais alargada do problema de escalonamento de máquinas paralelas (PM) por generalizar o problema ao considerar *setups*. Com esta introdução, o problema de escalonamento de máquinas paralelas torna-se mais complexo uma vez que sem os *setups* apenas existe o problema de alocação e a adição de *setups* incrementa o nível de decisão de sequenciação.

Em (Vallada & Ruiz, 2011) é estudado um algoritmo genético (metaheurística). O critério de otimização utilizado é a minimização do *makespan*. Além de o comparar com outros algoritmos genéticos (e comprovar a sua superioridade), este é comparado com uma versão adaptada do modelo de programação inteira (MIP) de (Guinet, 1993), com o intuito de se compreender o quão afastadas as soluções do AG se encontra da solução ótima do MIP, o que se provou não ser estatisticamente diferente, validando assim o seu desempenho. Posteriormente (Avalos-Rosales et al., 2015) demonstra uma MILP eficiente capaz de resolver instâncias até 60 tarefas e 5 máquinas.

Além das várias formulações inteiras mistas, o artigo mencionado propõe uma linearização da *makespan* que acelera o processo de solução. (Tran et al., 2016) utiliza o MILP anteriormente referido como problema mestre e propõe 2 métodos baseados em decomposição exata, o que revela capacidade para lidar com instâncias de 120 tarefas e 8 máquinas. Recentemente, em (Fanjul-Peyro et al., 2019), ficou provada que a melhoria de eficiência de um modelo se pode refletir pelo solver utilizado uma vez que o mesmo modelo de (Avalos-Rosales et al., 2015) consegue resolver instâncias de 200 tarefas e 4 máquinas. Além disso, também são melhorados modelos MILP (adicionando restrições baseadas em adaptações de restrições de eliminação de *subtour*) o que permite resolver instâncias de 400 tarefas e 4 máquinas. Por fim, neste mesmo artigo, também se melhora o proposto em (Tran et al., 2016) ao substituir o modelo MILP pelos propostos em (Fanjul-Peyro et al., 2019), o que permite resolver problemas de 1000 tarefas e 8 máquinas com desvios relativos dos limites inferiores abaixo de 0,8%.

2.3. Recursos

Num ambiente industrial, para o processamento de um determinado produto, as máquinas não são o único recurso que devemos ter em consideração. Normalmente, além de máquinas, são necessários outros recursos adicionais, nomeadamente: operadores de máquinas, moldes, gabaritos, dispositivos de fixação, ferramentas (Slowinski, 1980; Ventura & Kim, 2000).

Não considerar estes recursos é restritivo pois a não disponibilidade dos mesmos pode comprometer o funcionamento normal da fábrica, visto que também estes recursos são limitados. Além de os considerar, dever-se-á ter em atenção que estes recursos adicionais podem variar conforme a necessidade de cada máquina (Villa et al., 2018).

Segundo (Błażewicz et al., 2004), os recursos adicionais apenas podem ser considerados recursos de processamento se forem utilizados em conjunto com a máquina. Caso contrário, são considerados recursos *input-output*.

Do ponto de vista de (Slowinski, 1980) os recursos adicionais podem ser classificados relativamente à sua renovabilidade (limitação de uso do recurso) e quanto à divisibilidade. Assim, quanto à renovabilidade, identifica-se:

- Recurso renovável – recurso que pode ser utilizado numa tarefa e, novamente, na tarefa seguinte. Apenas existe limitação deste recurso em cada momento. Exemplos: operadores, máquinas, equipamentos, energia;
- Recurso não renovável – recurso que ao ser utilizado numa tarefa, fica indisponível para a seguinte. É preciso considerar a limitação total do recurso visto que ao ser utilizado, se esgota. Exemplos: matérias-primas, combustível, dinheiro;
- Recurso duplamente limitado – é, simultaneamente, renovável e não renovável.

Relativamente à divisibilidade de recursos:

- Recurso discreto – recurso que pode ser alocado em quantidades discretas a partir de um conjunto finito de alocações possíveis;
- Recurso contínuo – recurso que pode ser alocado em quantidades arbitrárias, dentro de um determinado intervalo.

Os recursos também poderão ser classificados como dinâmicos quando não permanecem fixos ao longo do período determinado.

2.3.1. Programação de Máquinas Paralelas com Recursos Adicionais

Os problemas de escalonamento de máquinas paralelas que consideram recursos são classificados como NP-hard. A inclusão de recursos no problema torna-o mais complexo, deixando de ser um problema apenas de alocação e sequenciação. Isto acontece porque não é possível garantir que o fluxo de trabalho numa máquina seja contínuo (podem existir esperas por falta de recursos). Assim, além de conhecermos a máquina onde uma tarefa vai ser processada bem como a ordem nessa mesma máquina, é necessário saber exatamente o momento temporal em que se reúnem condições para o processamento de determinada tarefa (Reklaitis, 1996) – a este terceiro nível de decisão dá-se o nome de tempo (ou timing).

Sumariando, este tipo de problemas lidará com três diferentes subproblemas:

- Alocação: afeta-se as tarefas/recursos às máquinas
- Sequenciação: ordena-se tarefas/recursos para execução
- Tempo: determina-se, especificamente, o início e fim de cada tarefa/recurso

Além disso, o aumento do número de recursos adicionais torna o problema ainda mais complexo uma vez que tem de se confirmar que para cada momento, se utilizam os recursos dentro dos limites de disponibilidade (Pinto & Grossmann, 1997).

Caso estejamos perante uma realidade de máquinas paralelas dedicadas, é possível simplificar o problema uma vez que o subproblema de alocação se extingue, conhecendo-se à priori a máquina em que cada tarefa será executada.

O PMR tem sido estudado nas últimas décadas. Nomeadamente em (J. Błażewicz et al., 1987) encontramos um modelo para resolver esta questão para máquinas idênticas e cuja função objetivo é a minimização do tempo médio de fluxo.

No artigo de (Fanjul-Peyro et al., 2017) é apresentado um novo modelo para resolver este problema com máquinas não relacionadas, baseando-se no *Strip-Packing*. Devido ao tamanho das instâncias, os autores decidiram utilizar 3 diferentes heurísticas matemáticas:

- *Machine-assignment fixing* – o problema é dividido em 2 subproblemas: alocação e sequenciação. Ao resolver o problema apenas de alocação (PM), transmite-se essa informação ao modelo PMR que apenas resolverá a sequenciação;
- *Job-machine reduction* – o número de máquinas em que uma tarefa pode ser executada é reduzido tendo por base a exclusão das máquinas onde o tempo de processamento de uma tarefa é maior;
- *Greedy-based fixing* – o problema é resolvido por fases, isto é, seleciona algumas tarefas e resolve o problema com as mesmas e, posteriormente, seleciona outras tarefas e resolve o problema com essas (não desconsiderando as primeiras atribuições). Este processo repete-se até que todas tenham sido estudadas.

Conclui-se que estas estratégias, comparativamente com os modelos IP, são bastante competitivas para pequenas instâncias (até 12 tarefas) mas destacam-se de forma evidente para instâncias de média dimensão (até 30 tarefas). No entanto, as heurísticas matemáticas apresentadas não se mostram muito eficientes para instâncias de maior dimensão (50 tarefas).

Num artigo mais recente, (Fleszar & Hindi, 2018), é proposto um novo modelo MILP, com o objetivo de minimizar a *makespan*, capaz de resolver instâncias até 1000 tarefas mas com apenas 2 máquinas não relacionadas, obtendo boas soluções – gap médio de 1,77% com CPU médio de 15 segundos. Na eventualidade de ser necessário mais de 2 máquinas, utiliza-se uma heurística de 2 estágios que consiste na utilização do MILP para calcular o limite inferior (*lower bound*) e resolver o problema no seu estado primário (PM) onde só se considera alocação, enquanto o CP faz a sequenciação e atribuição de recursos. Para este método obteve-se um gap médio de 0,64% e um CPU médio de 38 segundos. Caso não se encontre a solução ótima, o autor propõe que se use a solução proveniente da heurística de 2 estágios como *warm-start* para um modelo CP completo de PMR. Segundo o autor, estas propostas superam todos os modelos encontrados na literatura para o mesmo problema.

(Edis & Ozkarahan, 2012) considera-se um problema de escalonamento de máquinas paralelas com recursos e restrições de elegibilidade. Os autores desenvolvem um modelo de programação inteira, com o objetivo de minimizar a *makespan*, no entanto, este modelo apresentou dificuldades em resolver problemas com maiores instâncias, obtendo gaps elevados (gap máximo 50,82%) e um problema sem solução. Em prol de obter resultados mais eficientes, os autores propõem dividir o problema em 2 subproblemas: alocação e sequenciação/tempo. Esta nova estratégia será testada de duas formas: (primeira) com um modelo IP para cada uma das fases e (segunda) com um modelo IP para a fase de alocação e um modelo CP para a fase de sequenciação/tempo. Os resultados permitiram concluir que estas abordagens obtêm melhores resultados (especialmente nas grandes instâncias) nunca ultrapassando os 3% de gap, sendo que IP/IP tem melhor desempenho com um maior número de operadores e o IP/CP quando o número de operadores é mais reduzido.

2.3.2. Programação de Máquinas Paralelas com Recursos Flexíveis

A maioria dos problemas considera os tempos de processamento das tarefas conhecidos e fixos ao logo do tempo. Na prática, poderão existir situações em que a quantidade e combinação de recursos pode fazer variar o tempo de execução da tarefa (Daniels et al., 1996). A situação descrita representa o problema de máquinas paralelas com recursos flexíveis (PMFRS).

Este tipo de problema pode ser dinâmico ou estático. A principal diferença é que no problema dinâmico o recurso flexível pode ser alocado (e realocado) de forma livre em todas as máquinas ao longo do horizonte temporal da produção, enquanto no problema estático o recurso flexível pode ser alocado livremente entre as máquinas, mas esta atribuição deve permanecer fixa por um determinado período. Em (Edis & Oguz, 2012), a versão dinâmica é caracterizada como sendo mais realista. Apesar de este problema já conhecer a alocação das tarefas às máquinas, a versão dinâmica é complexa pois inclui 3 subproblemas: alocação dinâmica dos recursos, identificação da sequência de processamento das tarefas nas máquinas e a definição concreta dos horários de execução (Daniels et al., 1996). (Daniels et al., 1999) estudou a versão deste problema com máquinas não relacionadas, isto é, sem a previa alocação de tarefas às máquinas.

Em (Daniels et al., 1996) é realizado um paralelismo entre o PMR e o PMFRS, sendo que o autor classifica o PMR como um caso especial do PMFRS considerando que a possibilidade do tempo de processamento alterar com o número de recursos alocados é o único fator que os separa. No entanto, se considerarmos o PMR para máquinas não relacionadas, existem outros fatores em PMFRS, como a presença de máquinas dedicadas e de apenas um recurso adicional, também divergentes. Seguindo esta lógica, UPMFRS (problema de Escalonamento de Máquinas Paralelas Não Específicas com Recursos Flexíveis) é mais próximo a PMR com máquinas não relacionadas uma vez que as máquinas em que as tarefas serão executadas não são especificadas à priori (Edis et al., 2013).

(Daniels et al., 1996) explora pela primeira vez o conceito de PMFRS e apresenta formulações matemáticas para cada uma das versões do problema: estático (algoritmo polinomial) e dinâmico (modelo exato - com o objetivo de minimizar a *makespan*). Este problema foi estudado para instâncias até 80 tarefas, 4 máquinas e 6 recursos. O principal objetivo do artigo foi demonstrar que o desempenho operacional poderia ter melhorias significativas através da utilização de recursos flexíveis, sendo particularmente benéfica quando os recursos são escassos e os tempos de processamento são sensíveis à atribuição de recursos adicionais.

Já em (Daniels et al., 1999), para resolver o problema UPMFRS estático, é apresentado um MILP específico para a versão estática do problema (a decisão de alocação dos recursos permanecerá fixa até ao final do período) mas, devido à complexidade do problema, os autores optam por desenvolver 2 heurísticas: *Decomposition* (DH) e *Tabu-Search* (TBH). Estas heurísticas permitiram resolver instâncias até 50 tarefas, 5 máquinas e capacidade do único recurso de 7 unidades, sendo que a heurística com melhores resultados foi a *Tabu-Search*. Em contrapartida, em (Grigoriev et al., 2007) já nos encontramos perante uma versão dinâmica (a decisão de alocação dos recursos pode variar ao longo do horizonte temporal). O modelo ILP propõe também que, além da influência da máquina, também o número de recursos adicionais alocados (neste caso em particular: trabalhadores) possa influenciar o tempo de processamento de uma tarefa.

(Ólafsson & Shi, 2000) estendeu e reformulou o problema PMFRS dinâmico, desenvolvendo uma heurística *Nested Partitions* para a sua solução. Este é um algoritmo combina *local search* e *global search*. Inicialmente aplica-se o *global search* para identificar as sub-regiões que parecem ser as mais promissoras. Seguidamente segue-se, para cada iteração cada um dos seguintes 4 passos:

- *Partitioning*: dividir a região atualmente mais promissora em Q sub-regiões e agregar as restantes regiões;
- *Random sampling*: para cada uma das sub-regiões e região circundante agregada, criar uma amostra aleatória de soluções. É obrigatório que o escalonamento de cada solução selecionada tenha probabilidade positiva;
- *Estimating the promising index*: com as amostras de cada sub-região e região circundante, estima-se o índice promissor de cada sub-região. O índice usado foi a melhor *makespan*;
- *Backtracking*: Se uma das sub-regiões tiver o melhor índice promissor estimado, essa passará a ser a região mais promissora na próxima iteração.

Os resultados numéricos permitem compreender que é possível obter escalonamentos de alta qualidade por este método. Além disso, embora este método tenha semelhanças ao *branch-and-bound*, não corremos o risco de ter problemas de memória além de não ser tão

computacionalmente intensivo (uma vez que não requer um limite inferior e é apenas guiado por um limite superior de desempenho, o índice promissor).

Em (Edis & Oguz, 2012) são estudadas as versões estáticas e dinâmicas de PMFRS e UPMFRS. O autor propõe para algumas extensões para o modelo PMFRS dinâmico proposto em (Daniels et al., 1996) e apresenta novos modelos de programação inteira para problemas UPMFRS estáticos e dinâmicos, ambos com o objetivo de minimização de *makespan*. Além dos modelos IP, os autores decidiram desenvolver uma abordagem híbrida (modelo IP e *Constraint Programming*) que consistia na utilização do modelo IP relaxado de (Grigoriev et al., 2005) para alocação de recursos e posteriormente CP para definir a sequência de execução das tarefas em cada máquina. Para o PMFRS, foi possível obter um gap médio de 0,82%, em que 66,1% dos problemas é encontrada a solução ótima – em soluções até 20 tarefas, 5 máquinas e 7 recursos (CPU médio de 10seg), gap médio inferior a 0,31%, em que cerca de 70% dos problemas é encontrada a solução ótima – em soluções de 50 tarefas e até 10 máquinas e 20 recursos (CPU médio de 20seg) e gap médio inferior a 0,67%, em que cerca de 50% dos problemas é encontrada a solução ótima – em soluções de 100 tarefas e até 10 máquinas e 20 recursos (CPU médio de 60seg). No UPMFRS obtém-se resultados idênticos, que destacam a boa performance, principalmente, nas instâncias grandes (gap médio inferior a 1,43%, sendo encontrada a solução ótima em 20% dos problemas com 100 tarefas, com CPU médio de 75seg). Comparativamente aos modelos IP, para pequenas e médias instâncias o modelo IP consegue atingir, praticamente sempre, soluções ótimas (no entanto, atinge tempos computacionais de 1 hora, valores muito superiores à média máxima de IP-CP) mas o IP-CP destaca-se para grandes instâncias, produzindo melhores resultados.

Em (Fanjul-Peyro, 2020) também existem recursos flexíveis mas numa ótica diferente da apresentada por (Daniels et al., 1996) – os recursos (neste caso, operadores) podem ser flexíveis ao nível das tarefas que desempenham, isto é, tanto poderão ser utilizados em atividades de processamento como de *setup*, conforme a sua disponibilidade. Este artigo será explorado em 2.4.

2.4. Programação de Máquinas Paralelas com *Setups* Dependentes da Sequência e Recursos Adicionais

PMRS é a sigla atribuída aos problemas de escalonamento com restrições de recursos e *setup*s. Este problema define-se como um conjunto de n tarefas que serão executadas por um conjunto de m máquinas paralelas, conhecendo-se o tempo de processamento da tarefa j na máquina i que será de p_{ji}, o tempo do *setup* entre a tarefa k e a tarefa j na máquina i de s_{ikj} e os seus requisitos em recursos. Os recursos existentes podem ser de vários tipos $R = \{R_1,\ R_2, \dots, R_u\}$, conhecendo-se a disponibilidade em quantidades $b_1,\ b_2, \dots, b_u$, respetivamente (Edis et al., 2013).

Casos Práticos

O PMSR surge como uma versão mais genérica da PMS e PMR, considerando, simultaneamente *setups* e recursos adicionais.

(Ozer & Sarac, 2019) estudam o problema PMSR, contemplando *setups* e recursos, no entanto, neste caso, os recursos adicionais são apenas de processamento, isto é, não são considerados outros tipos de recursos (p.e. recursos de *setup*). No entanto, são consideradas restrições de

elegibilidade e cópias de recursos partilhados (IPMS-SMS). Saliente-se que se trata de máquinas idênticas e o objetivo é minimizar o tempo total ponderado da execução das tarefas. O autor propõe um MILP e uma heurística matemática composta por um modelo de programação inteira mista (MILP) e um modelo baseado no algoritmo genético (GA). No AG clássico, as soluções são codificadas nos cromossomas e contém todas variáveis de decisão, no entanto, nesta metaheurística, o AG apenas conterá a variável x_{jkl} uma vez que só resolverá a componente de alocação e sequenciação do problema. O modelo matemático terá os valores de x_{jkl} como parâmetro e irá determinar o tempo de início, conclusão e espera das tarefas. O valor da função objetivo do MILP é o valor da aptidão do cromossoma (ou *fitness*) que será devolvido ao mesmo.

O MILP mostra excelentes resultados para as instâncias de menor dimensão (solução ótima é encontrada quase sempre), no entanto, para as instâncias de média/grande dimensão (40 e 100 tarefas) é notório o melhor desempenho da heurística matemática.

(Bektur & Saraç, 2019) aborda o problema PMSR com máquinas não relacionadas considerando que os tempos de *setup* apenas dependem da sequência e não da máquina (uma vez que partilham o mesmo servidor) e restrições de elegibilidade de máquinas. O autor define que o objetivo será minimizar o atraso ponderado total de cada tarefa e para a resolução deste problema propõe duas vertentes: um MILP e duas diferentes metaheurísticas (*Tabu Search* e *Simulated Annealing*). Além disso, o autor inclui uma regra para a obtenção de soluções iniciais que se relaciona com o custo dos atrasos: ATCS. Para um tempo máximo de 18000 segundos, o MILP demonstrou só conseguir atingir soluções ótimas para problemas de 2 máquinas e 10 tarefas. Relativamente às metaheurísticas, estas são capazes de resolver grandes instâncias (10 máquinas e 100 tarefas) com bons resultados, sendo que as que utilizam a regra ATCS tem melhor performance (erro percentual, TS: 0,28% e SA: 5,58%) face às que não utilizam (erro percentual, TS: 13,60% e SA: 9,20%). Conclui-se que o algoritmo *Tabu Search,* em conjunto com a regra ATCS, produzem os melhores resultados.

Em (Fanjul-Peyro, 2020), o autor propõe um modelo para resolver o problema UPMSR, contando com diferentes tipos de recursos: recursos de processamento (específicos para o processamento da tarefa na máquina), recursos de *setup* (necessários à atividade de *setup*) e recursos partilhados (recursos que tanto podem ser utilizados em *setup* como em processamento). O indicador de desempenho utilizado foi a minimização da *makespan*. Neste artigo o autor propõe dois MILP: um baseado no *Strip-Packing* e outro indexado ao tempo (*Time Index*), sendo que o MILP baseado em *Strip-Packing* é o que apresenta melhores resultados. Este MILP é também proposto para a resolução do problema PMR com máquinas não relacionadas e comparado com os resultados obtidos com o MILP proposto por (Fanjul-Peyro et al., 2017) e por um outro MILP, também desenvolvido pelo autor, que consiste numa reformulação do modelo anteriormente citado – o que permitiu concluir que o modelo geral permite melhores resultados que o particular, obtendo um gap de 7% (que se destaca dos outros gaps médios de 385% e 174%).

Além disso, para resolver maiores instâncias, também um algoritmo trifásico (TPhA) é sugerido, dividindo a resolução em três principais etapas: alocação, sequenciação e tempo, do seguinte modo:

- Realiza-se a atribuição com um modelo ILP, verificando-se se a solução obtida é melhor que melhor solução anterior. Caso seja melhor, verifica-se a solução é ótima;

- Se a solução for ótima, as etapas de sequenciação e timing são resolvidas de forma integrada (para obtermos uma solução ótima completa). Caso contrário, primeiro realiza-se a etapa de sequenciação e depois de timing (quando realizado individualmente, utiliza-se um modelo CP para a execução do timing);
- Caso a solução obtida no MILP da alocação seja, simultaneamente, ótima e não seja melhor que a melhor solução anterior,

Este algoritmo produz bons resultados, resolvendo instâncias até 400 tarefas e 8 máquinas com gaps inferiores a 2,7% (para grandes instâncias). Além disso, TPhA tem melhor performance (gap médio de 6,52%) que o modelo MILP *Strip-Packing* também proposto no estudo (gap médio de 9,15%).

O estudo de (Yepes-Borrero et al., 2020) também considera recursos e *setups*, no entanto, neste caso os recursos considerados são exclusivamente para os *setups*. Assim, o autor recorre ao algoritmo GRASP para responder ao problema e minimizar a *makespan*. São então propostas três metaheurísticas, seguindo duas diferentes abordagens: a desconsiderar a informação sobre recursos adicionais (2 metaheurísticas) ou considerar a informação sobre recursos adicionais (1 metaheurística). Os autores propões também um mecanismo de reparação que verifica se o limite de recursos utilizado é excedido em algum momento e, caso se confirme, são adiadas as atividades de *setup*. Além disso, é sugerido um algoritmo de *local search* para a inicialização da solução. O algoritmo foi testado em instâncias até 250 tarefas, 30 máquinas e 4 recursos de *setup*. Foi possível compreender que os 2 algoritmos propostos para a primeira abordagem e o algoritmo proposto para a segunda abordagem produzem resultados idênticos para instâncias pequenas e médias mas, em instâncias grandes, a segunda abordagem (considerar a informação sobre recursos adicionais) destaca-se obtendo um gap de 0,19%, face aos gaps de 39,97% e 69,66% dos outros algoritmos da primeira abordagem. Além de se concluir que a segunda abordagem tem um impacto muito positivo em grandes instâncias, é reconhecido que também o *local search* inicial contribui significativamente para a eficiência dos algoritmos.

Uma abordagem multiobjectivo é proposta por (Yepes-Borrero et al., 2021), procurando-se minimizar a *makespan* e o número de recursos adicionais. Neste artigo é proposto um algoritmo, nomeado T-RIPG, que resulta da adaptação do algoritmo RIPG proposto por (Bektur & Saraç, 2019). Este algoritmo possui 6 fases:

- Inicialização – utilização de um modelo GRASP (*Greedy Randomized Adaptive Search Procedure)* para gerar boas soluções iniciais (balanceadas para ambos os objetivos de *makespan*);
- Fase gulosa – a partir da solução inicial, destroem-se e reconstroem-se novas soluções;
- Temporização/ reparação – esta etapa não é obrigatória e acontecerá mediante uma probabilidade p. Caso ocorra, define-se aleatoriamente, dentro de um determinado intervalo, o número máximo de recursos. A diminuição do número de recursos através do adiamento do início de uma atividade de *setup*, mesmo que isso possa significar aumento da *makespan*;
- Seleção – dentro do conjunto de trabalho de soluções, escolhe-se uma para continuar a ser explorada nas fases seguintes;
- *Local search* – posterior à fase de reparação, na etapa seleção, define-se a solução que será processada pelo *local search*. O processo será repetido enquanto se identifiquem melhorias na solução;

- Reiniciar – são guardadas as soluções obtidas e retorna-se à fase inicial (Inicialização), criando um novo conjunto de soluções iniciais com maior variabilidade.

Este algoritmo demonstra ser melhor que os restantes algoritmos estudados pelo respetivo autor, conseguindo produzir soluções até 250 tarefas e 30 máquinas.

Em (Yunusoglu & Yildiz, 2022) é estudado o problema PMSR para máquinas não relacionadas, com o objetivo de minimizar o tempo máximo de término das tarefas. Neste caso são também consideradas restrições de elegibilidade da máquina e datas de lançamento. Assim, é proposto um modelo de *Constraint Programming* (CP) agregado a duas estratégias de ramificação para o mesmo modelo (permitindo orientar a procura da solução): ordenação de variáveis (seleciona as variáveis a ramificar) e ordenação de valores (ordenação dos valores possíveis de serem atribuídos à variável escolhida). Segundo os autores, este modelo CP permite resolver instâncias até 60 tarefas, 8 máquinas e 2 recursos, superando modelos de programação inteira bem como metaheurísticas do estado da arte, atingindo um GAP médio de 15,52%.

2.5. Heurísticas Matemáticas

Considerando a complexidade do problema estudado, destaca-se a importância de utilizar heurísticas matemáticas para permitir um melhor desempenho por parte do modelo matemático. Este método combina heurísticas/metaheurísticas com modelos de programação matemática, permitindo obter as vantagens de cada um deles.

Segundo (Puchinger & Raidl, 2005) este tipo de mecanismo de resolução pode ser classificado em duas categorias:

- Combinações colaborativas: o algoritmo exato e a metaheurística trocam informações, mas não fazem parte um do outro. Os algoritmos podem ser executados sequencialmente, em paralelo ou entrelaçados;
- Combinações Integrativas: um método (o método escravo) é modificado de maneira a incorporar o outro (o método mestre). O algoritmo mestre pode ser tanto o algoritmo exato como o heurístico, sendo obrigatório existir, pelo menos, um escravo.

Uma vez que esta é uma abordagem recente, não existem muitos artigos na literatura para o problema das máquinas paralelas.

(Dang et al., 2021; Fanjul-Peyro et al., 2017; Ozer & Sarac, 2019) são exemplos de combinações colaborativas. Por exemplo, em (Dang et al., 2021) utiliza-*se heurísticas matemáticas para* resolver um problema de escalonamento de máquinas paralelas com substituição de ferramenta. A estratégia seguida passou por dividir o problema em 3 subproblemas: alocação, sequenciação e substituição de ferramentas, sendo os primeiros 2 (alocação e sequenciação) resolvidos por uma metaheurística conhecida, o Algoritmo Genético, que posteriormente atribuiu esta parte da solução ao MILP que resolveu o 3º subproblema (substituição de ferramentas). Cada um dos artigos mencionados concluíram que as heurísticas matemáticas e outros modelos matemáticos têm performance idêntica para pequenas instâncias, mas que se destacam verdadeiramente em instâncias médias e grandes. (Fanjul-Peyro et al., 2017) e (Ozer & Sarac, 2019), previamente explorado nos capítulos 2.3.1. e 2.4., respetivamente, são casos de sucesso da abordagem.

Na literatura não foi possível identificar heurísticas matemáticas integrativas para o problema das máquinas paralelas. No entanto, esta abordagem já foi explorada para a resolução do problema de escalonamento de máquina única, consulte (Della Croce et al., 2014; Henrique et al., 2014; Sioud et al., 2012).

Apesar de existirem várias estratégias possíveis no âmbito das heurísticas, dar-se-á particular importância ao *warm-start* neste subcapítulo.

2.5.1. Aplicação de *Warm-start*

Warm-start baseia-se no fornecimento de uma solução inicial de boa qualidade ao sistema, utilizando outros algoritmos para a resolução do mesmo problema (ou de apenas parte do problema). Esta solução permite que o sistema faça cortes na árvore *branch & bound* o que possibilita uma redução de tempo de execução, tornando o modelo mais eficiente (Heinz et al., 2022; M. Pour et al., 2018).

Desta forma, com a aplicação do *warm-start*, será possível diminuir os tempos computacionais elevados, permitindo (ou simplesmente acelerando) o processo de obtenção de solução ótima e reduzindo o risco de erros de memória (devido à redução de iterações necessárias).

Casos Práticos

Em (Heinz et al., 2022) é estudado um caso de máquinas paralelas idênticas não dedicadas com *setups* dependentes da sequência que são executados por servidores. O objetivo passa por minimizar a *makespan*. Neste trabalho é utilizado um modelo de *Constraint Programming* para encontrar a solução ótima, utilizando-se duas heurísticas para fornecer uma solução inicial viável, melhorando assim o seu desempenho, possibilitando o alcance de soluções para problemas de 20 máquinas e 500 tarefas em 10 segundos (com uma média de *makespans* superiores em 5% aos limites inferiores calculados). Uma vez que o âmbito deste subcapítulo é o estudo do *warm-start* na literatura, a análise deste artigo focar-se-á nas heurísticas utilizadas para o efeito. Este artigo propõe 2 diferentes algoritmos heurísticos construtivos: LOSOS (*Locally Optimal Selection of Setups*) e ROSOL (*Resolution of Setup Overlaps Lazily*).

- Em LOSOS, inicialmente atribuem-se, de forma aleatória, as tarefas iniciais de cada máquina, registando-se os horários em que os servidores terão de trabalhar. Repetidamente, enquanto existirem tarefas por alocar: identifica-se a máquina com menor tempo de programação e atribui-se uma nova tarefa, verificando-se a disponibilidade dos servidores para a realização de *setup* e atribuindo-se aquele que fica disponível mais rapidamente. No final é realizada uma otimização no cronograma;
- Em ROSOL, também existe a atribuição aleatória da primeira tarefa a cada máquina. Enquanto existirem tarefas por atribuir, identifica-se a máquina com a menor programação e é-lhe atribuída uma tarefa (sempre sem considerar os servidores). Posteriormente, otimiza-se o cronograma para extinguir potenciais conflitos. De seguida são analisadas as disponibilidades dos servidores ao longo do cronograma, do início para o fim, e sempre que se verifique que o número de servidores utilizados em simultâneo excede o limite, adiam-se os *setups* até que pelo menos um servidor esteja livre. No fim deste processo, realiza-se

a otimização do cronograma, novamente. Neste algoritmo, quando se identificam um excesso de servidores em simultâneo, e é necessário adia-se o *setup* de uma das tarefas, calcula-se o coeficiente de tolerância (diferença entre o comprimento do cronograma e o *makespan* atual mais o comprimento de configuração a ser executado atualmente naquela máquina) para todas as máquinas envolvidas no conflito e adiam-se as tarefas das máquinas com menor coeficiente.

O próprio artigo sugere algumas melhorias para os algoritmos propostos, nomeadamente as seguintes:

- Não tornar a atribuição da primeira tarefa aleatória. Sugere-se que se procurem as sequências de tarefas com menores *setups* e destas, selecionamos as que tem tempos de processamento menores;
- Identificar a máquina com o maior cronograma e compreender se é possível mover a sua última tarefa para outra máquina, diminuindo a *makespan*. Caso seja possível, repetir o processo enquanto se verificar a redução da *makespan*;
- Alterar as tarefas finais entre pares de máquinas, procurando soluções que diminuam o comprimento do par de máquinas mas, especialmente, soluções que permitam diminuir o *makespan*.

A performance das duas heurísticas é muito similar, com ROSOL a mostrar-se ligeiramente melhor, principalmente para instâncias com menos servidores.

Flowshop é a tipologia de problema abordada em (de Abreu & Fuchigami, 2022). Neste artigo os autores implementam diferentes modelos de programação linear inteira (MILP) e, posteriormente, testam o impacto de fornecer uma solução inicial a cada um dos modelos (*warm-start*), a partir de três diferentes algoritmos heurísticos conhecidos: SPL (*shortest processing time),* LPL *(longest processing time)* e uma adaptação de NEH (*Nawaz, Enscore, and Ham*) – NEH IW. Este estudo contempla 240 instâncias de dimensões até 500 tarefas em 20 máquinas e permitiu concluir que, com a implementação do *warm-start*, o número de soluções ótimas e viáveis aumentou e que os resultados globais do modelo melhoraram, obtendo-se resultados mais robustos. Além disso, para este caso, o LPT foi a heurística que se mostrou mais eficaz, seguido do SPT, do modelo puro e, por último, do NEHIW.

(Vahedi-Nouri et al., 2022) estudou o comportamento de modelos MILP e CP, bem como versões melhoradas dos mesmos, I-MILP e I-CP, para o planeamento da mão de obra e produção num ambiente RMS (*Reconfigurable Machine Tool*). Ao compreender que o I-MILP permitia obter o melhor desvio percentual relativo (RPD), o autor decidiu usar parcialmente essa solução como *warm-start* para o modelo I-CP. Este modelo híbrido permitiu obter os melhores resultados na maior parte das instâncias, reduzindo, em média, o RPD de 4,44 (I-MILP) para 0,91, além de que o fez com tempos computacionais muito menores.

Em (Fleszar & Hindi, 2018) são exploradas diferentes estratégias de heurísticas matemáticas. Primeiro é utilizado um MILP para calcular o *lower bound* e resolver o problema de alocação. Posteriormente são fornecidos estes dados ao CP que terá a função de sequenciar as tarefas e atribuir os recursos. Caso o algoritmo não encontre a solução ótima, o autor propõe que se use a solução proveniente da heurística de 2 estágios como *warm-start* para um modelo CP completo de PMR. Com um limite computacional de 60 segundos, a 2ª abordagem permitiu obter soluções

ótimas em todas as instâncias com 8 e 12 tarefas, evidenciando um desempenho superior comparativamente à abordagem de duas fases. Também a introdução do LB e do UP demonstraram claras melhorias no desempenho do modelo. Os autores também compararam sua estratégia com os algoritmos de (Fanjul-Peyro et al., 2017) e obtiveram melhores soluções e com menor tempo computacional para instâncias até 30 tarefas. A estratégia de *warm-start* alcançou um gap médio de 0,10% e um gap máximo de 4,14%, enquanto os algoritmos de (Fanjul-Peyro et al., 2017) obtiveram um gap médio de 1,35% e um gap máximo de 22,03%. Adicionalmente, com um tempo limite de 300 segundos, a estratégia de *warm-start* encontrou soluções ótimas em todas as instâncias até 16 tarefas, tendo um gap médio de 0,46% para todos os testes. No caso específico de 500 e 1000 tarefas, os gaps médios obtidos foram 1,13% e 1,22%, respetivamente.

Por outro lado, em (M. Pour et al., 2018) procura-se fazer o escalonamento da manutenção preventiva de um sistema ferroviário. Nesta problemática são consideradas múltiplas restrições reais como dependências temporais, horários da tripulação, divisão de tarefas pelos dias, competências da tripulação, entre outros. No artigo é proposta uma solução híbrida que se traduz na utilização de um algoritmo CP de forma a obter uma solução inicial que vai alimentar o MILP. Devido aos menores esforços computacionais do MILP, esta estrutura híbrida permite que se gerem bons resultados para 8 semanas (uma melhoria significativa face às 2 semanas da solução inicial – apenas MILP). Além disso, também se torna evidente que receber uma boa solução inicial é impactante ao nível do gap, reduzindo valores entre 60-80% para 3-23%.

Deste modo é possível concluir que na literatura a aplicação da técnica de *warm-start* revela um impacto positivo, permitindo a obtenção de melhores soluções e/ou menores tempos computacionais. Além disso, esta estratégia revela ser particularmente importante em problemas muito complexos e com instâncias de maior dimensão, o que se enquadra no contexto trabalhado.

2.6. Análise de Indicadores de Desempenho

Os indicadores de desempenho são métricas que permitem orientar o modelo na busca pela melhor solução, sendo essencial que se reflitam o objetivo da problemática. Na programação matemática, os indicadores de desempenho irão variar conforme a tipologia de problema e as particularidades de cada caso.

O estudo dos indicadores de desempenho para o problema das máquinas paralelas revela que o indicador mais utilizado é a *makespan*. No entanto, foi também possível encontrar problemas que objetivavam a minimização dos atrasos totais ponderados bem como o tempo de conclusão ponderado das tarefas.

Além de funções objetivo com 1 único objetivo, também foram encontradas funções multi-objetivo. Os dois artigos nesta situação priorização a minimização da *makespan* e, apenas quando está fica garantida, procuram a minimização do consumo total de energia das máquinas em (Zhang et al., 2021) e a minimização do número de recursos utilizados (Yepes-Borrero et al., 2021).

Recorrendo à literatura de problemas de máquinas paralelas, construiu-se a Tabela 1 onde se escrutina o indicador de desempenho utilizado em cada caso de estudo analisado.

Tabela 1 – Análise e comparação de indicadores de desempenho

Caso de Estudo	Problema	Estratégia	Recursos Adicionais	Indicador de desempenho
(Heinz et al., 2022)	Máquinas paralelas idênticas	CP e heurística matemática	Sim, mas apenas recursos de *setup*.	Minimização *makespan*
(Soares & Carvalho, 2022)	Máquinas paralelas idênticas	Metaheurística	Sim	Minimização *makespan*
(Yunusoglu & Yildiz, 2022)	Máquinas paralelas não relacionadas	CP	Sim.	Minimização *makespan*
(Fleszar & Hindi, 2018)	Máquinas paralelas não relacionadas	MILP, CP e heurística matemática	Sim.	Minimização *makespan*
(Fanjul-Peyro et al., 2019)	Máquinas paralelas não relacionadas	MILP	Não.	Minimização *makespan*
(Fanjul-Peyro et al., 2017)	Máquinas paralelas não relacionadas	MILP, heurística matemática	Sim.	Minimização *makespan*
(Fanjul-Peyro, 2020)	Máquinas paralelas não relacionadas	MILP	Sim.	Minimização *makespan*
(Yepes-Borrero et al., 2020)	Máquinas paralelas não relacionadas	Metaheurística	Sim.	Minimização *makespan*
(Ólafsson & Shi, 2000)	Máquinas dedicadas	Heurísticas	Sim.	Minimização *makespan*
(Bektur & Saraç, 2019)	Máquinas paralelas não relacionadas	MILP e Metaheurística	Sim, mas apenas 1 recurso de *setup*	Minimização do atraso total ponderado
(Ozer & Sarac, 2019)	Máquinas paralelas idênticas	MILP	Sim.	Minimização do tempo de conclusão ponderado de cada tarefa
(Bitar et al., 2021)	Máquinas paralelas não relacionadas	MILP	Sim.	3 diferentes funções-objetivo: maximização do número de produtos concluídos no período temporal definido & minimização da soma ponderada dos tempos de conclusão & minimização do número de movimentações de recursos
(Zhang et al., 2021)	Máquinas paralelas não relacionadas	Algoritmo evolutivo combinatório (CEA)	Sim.	Multiobjetivo: minimização da *makespan* & minimização do consumo total de energia das máquinas
(Yepes-Borrero et al., 2021)	Máquinas paralelas não relacionadas	Metaheurística	Sim.	Multiobjectivo: minimização *makespan* & minimização nº recursos

2.7. Análise de *Solvers*

Os modelos matemáticos requerem que se recorra a um solver para a resolução do problema. Assim, é possível encontrar *solvers* de código aberto (*COIN-OR, SCIP*,…) ou *solvers* comerciais (*CPLEX, GUROBI*,…). Segundo (Cheng et al., 2022), GUROBI foi reconhecido como sendo o mais rápido MILP *solver.* Esta informação alinha-se com o contínuo trabalho desenvolvido por (Mittelmann, 2022) que afirma que desde a atualização de 2018, o GUROBI se destaca dos restantes.

A Tabela 2 reflete os resultados obtidos quando comparados diferentes *solvers* na literatura.

Apesar de não ser unanime, os estudos mencionados parecem revelar uma preferência consistente pelo *solver GUROBI*.

Tabela 2 – Casos de Estudo relativos ao desempenho de diferentes *Solvers*

Study case	Problema	Solvers	Melhor solver
(Ku & Beck, 2016)	*Job-shop*	*CPLEX, GUROBI, SCIP*	*CPLEX* e *GUROBI* similares. *CPLEX* ligeiramente melhor.
(Nguyen et al., 2013)	Máquina única	*GUROBI, GLPK, COIN-OR*	*GUROBI*
(Hatami et al., 2016)	Máquinas Paralelas	*GUROBI, CPLEX*	Não há clara preferência
(Fanjul-Peyro et al., 2019)	Máquinas Paralelas	*GUROBI, CPLEX*	*GUROBI*
(Fanjul-Peyro, 2020)	Máquinas Paralelas	*GUROBI, CPLEX*	*GUROBI*
(Wang et al., 2020)	Máquinas Paralelas	*GUROBI, CPLEX*	*GUROBI*
(Hatami et al., 2013)	*Flowshop*	*GUROBI, CPLEX*	*GUROBI*

2.8. Modelo de Referência

O trabalho desenvolvido centrar-se-á nos modelos *Strip-Packing* e *Time Index* propostos por (Fanjul-Peyro, 2020). Este subcapítulo contempla a definição de parâmetros e variáveis e os modelos desenvolvidos. Note-se que os parâmetros/variáveis associados exclusivamente a um dos tipos de modelos são definidos apenas junto a esses.

Assim, os parâmetros comuns necessários são:

- Um conjunto de *m* máquinas que podem processar as tarefas, denotado como $M = \{1, \dots, m\}$, indexado por i.
- Um conjunto de n tarefas a serem processadas, $N = \{1, \dots, n\}$, indexado por j, k.
- p_{ij} representa o tempo necessário para a máquina i processar a tarefa j.
- s_{ijk} representa o tempo necessário para efetuar o *setup* na máquina i entre as tarefas j e k.
- r_{max}^{P} é a quantidade máxima de recursos de processamento. Estes recursos são necessários especificamente para o processamento das tarefas e não podem ser usados nos *setups*, sendo denotados como recursos-P.

- r_{max}^{S} é a quantidade máxima de recursos de recursos de *setup*. Estes recursos são necessários especificamente para as operações de *setup* e não podem ser usados no processamento das tarefas, sendo denotados como recursos-S.
- r_{max}^{H} é a quantidade máxima de recursos não específicos que podem ser usados no processamento e nos *setups*. Estes recursos são denotados como recursos-H.
- r_{ij}^{P} são os recursos-P necessários para o processamento da tarefa j na máquina i.
- r_{ijk}^{S} são os recursos-S necessários para o *setup* entre a tarefa j e a tarefa k na máquina i.
- r_{ij}^{HP} são os recursos-H necessários para o processamento da tarefa j na máquina i. O processamento da tarefa j na máquina i pode necessitar r_{ij}^{P} de recursos-P e r_{ij}^{HP} de recursos-H.
- r_{ijk}^{HS} são os recursos-H necessários para o *setup* entre as tarefas j e k na máquina i. O *setup* entre as tarefas j e k na máquina i pode necessitar r_{ijk}^{S} de recursos-P e r_{ijk}^{HS} de recursos-H.
- *M* é uma constante suficientemente grande.

As variáveis de decisão comuns aos dois modelos são:

- $X_{ijk} = 1$ se a tarefa k é processada imediatamente a seguir à tarefa j na máquina i, $X_{ijk} = 0$ caso contrário.
- $Y_{ik} = 1$ se a tarefa k é processada na máquina i, $Y_{ik} = 0$ caso contrário.
- C_j é o tempo de conclusão da tarefa *j* (coordenada onde o processamento da tarefa *j* termina na dimensão temporal) (minutos).
- B_j é o tempo de conclusão do *setup* antes do processamento da tarefa *j* na máquina correspondente (coordenada onde o *setup* da tarefa *j* termina na dimensão temporal) (minutos).

As restrições apresentadas de seguida (1)-(12) são também comuns a ambos os modelos.

$$Min\ Z = C_{max} \tag{1}$$

$$\sum_{j \in N_0, k \in N, j \neq k} s_{ijk} X_{ijk} + \sum_{k \in N} p_{ik} Y_{ik} \leq C_{max}, i \in M \tag{2}$$

$$\sum_{k \in N} X_{i0k} \leq 1, i \in M \tag{3}$$

$$\sum_{i \in M} Y_{ik} = 1, k \in N \tag{4}$$

$$Y_{ik} = \sum_{j \in N_0, j \neq k} X_{ikj}, i \in M, k \in N \tag{5}$$

$$Y_{ik} = \sum_{j \in N_0, j \neq k} X_{ijk}, i \in M, k \in N \tag{6}$$

$$C_k - C_j + M\left(1 - X_{ijk}\right) \geq s_{ijk} + p_{ik}, i \in M, j \in N_0, k \in N, j \neq k \tag{7}$$

$$C_k \leq C_{max}, k \in N_0 \tag{8}$$

$$X_{ijk} \in \{0,1\}, Y_{ij} \geq 0, C_k \geq 0, C_0 = 0, j,k \in N, i \in M \quad (9)$$

$$B_k \leq C_k - \sum_{i \in M} Y_{ik} p_{ik}\,, k \in N \quad (10)$$

$$B_k - C_j + M\left(1 - \sum_{i \in M} X_{ijk}\right) \geq \sum_{i \in M} s_{ijk} X_{ijk}\,, j \in N_0, k \in N, j \neq k \quad (11)$$

$$C_k \geq \sum_{i \in M, j \in N_0} s_{ijk} X_{ijk} + \sum_{i \in M} Y_{ik} p_{ik}\,, k \in N \quad (12)$$

O MILP *Strip-Packing* carece da introdução de variáveis de decisão e restrições específicas.

- $R_j^P \in [0, r_{max}^P]$ é a coordenada do processamento da tarefa j na dimensão recursos-P (recursos específicos necessários para o processamento da tarefa j).
- $R_j^S \in [0, r_{max}^S]$ é a coordenada do *setup* da tarefa j na dimensão recursos-S (recursos específicos necessários para o *setup* que ocorre antes do processamento da tarefa j).
- $D_{jk}^P, E_{jk}^P, F_{jk}^S\ G_{jk}^S, U_{jk}^{HP}, U_{jk}^{HS}, V_{jk}^{HP}, U_{jk}^{HS}$ são variáveis binárias auxiliares com o valor = 1 se o valor da variável estudada nas restrições (tempo ou recursos) da tarefa k é maior que o valor da variável estudada (tempo ou recursos) da tarefa j. As letras S, P, HP e HS são usadas para indicar a dimensão em que as variáveis são usadas.

$$r_{max}^P \geq R_k^P \geq \sum_{i \in M} Y_{ik} r_{ik}^P\,, k \in N \quad (13)$$

$$r_{max}^S \geq R_k^S \geq \sum_{i \in M, l \in N} X_{ilk} r_{ilk}^S\,, k \in N \quad (14)$$

$$r_{max}^H \geq R_k^{HP} \geq \sum_{i \in M} Y_{ik} r_{ik}^{HP} \geq R_k^{HS} \geq \sum_{i \in M, l \in N} X_{ilk} r_{ilk}^{HS}\,, k \in N \quad (15)$$

$$C_k - C_j + M\left(1 - D_{jk}^P\right) \geq \sum_{i \in M} Y_{ik} p_{ik}\,, j,k \in N, j \neq k \quad (16)$$

$$R_k^P - R_j^P + M\left(1 - E_{jk}^P\right) \geq \sum_{i \in M} Y_{ik} r_{ik}^P\,, j,k \in N, j \neq k \quad (17)$$

$$D_{jk}^P + D_{kj}^P + E_{jk}^P + E_{kj}^P \geq 1, j,k \in N, j \neq k \quad (18)$$

$$B_k - B_j + M\left(1 - F_{jk}^S\right) \geq \sum_{i \in M, l \in N_0} X_{ilk} s_{ilk}\,, j,k \in N, j \neq k \quad (19)$$

$$R_k^S - R_j^S + M\left(1 - G_{jk}^S\right) \geq \sum_{i \in M, l \in N_0} X_{ilk} r_{ilk}^S\,, j,k \in N, j \neq k \quad (20)$$

$$F_{jk}^S + F_{kj}^S + G_{jk}^S + G_{kj}^S \geq 1, j,k \in N, j \neq k \quad (21)$$

$$C_k - B_j + M\left(1 - U_{jk}^{HP}\right) \geq \sum_{i \in M} Y_{ik} p_{ik}\,, j, k \in N, j \neq k \tag{22}$$

$$B_j - C_k + M\left(1 - U_{kj}^{HS}\right) \geq \sum_{i \in M, l \in N_0} X_{ilj} r_{ilj}^{S}\,, j, k \in N, j \neq k \tag{23}$$

$$R_k^{HP} - R_j^{HS} + M\left(1 - V_{jk}^{HP}\right) \geq \sum_{i \in M} Y_{ik} r_{ik}^{HP}\,, j, k \in N, j \neq k \tag{24}$$

$$R_j^{HS} - R_k^{HP} + M\left(1 - V_{jk}^{HS}\right) \geq \sum_{i \in M, l \in N_0} X_{ilj} r_{ilj}^{HS}\,, j, k \in N, j \neq k \tag{25}$$

$$U_{jk}^{HP} + U_{kj}^{HS} + V_{jk}^{HP} + V_{kj}^{HS} \geq 1, j, k \in N, j \neq k \tag{26}$$

No MILP Time Index, é necessário introduzir as seguintes variáveis específicas:

- $E_{ikt} = 1$ se k estiver em processamento na máquina i no instante t, $E_{ikt} = 0$ caso contrário.
- $F_{ijkt} = 1$ se k estiver em *setup* depois da tarefa j na máquina i no instante t, $F_{ijkt} = 0$ caso contrário.
- t_{max} é o horizonte do planeamento.

E_{ikt} e F_{ijkt} são variáveis de decisão e t_{max} é um parâmetro. Além equações comuns (1) – (12), é necessário adicionar as restrições (26)-(35).

$$\sum_{t \leq t_{max}} E_{ikt} \geq p_{ik} Y_{ik}\,, i \in M, k \ \in N \tag{27}$$

$$C_k \geq \sum_{i \in M} t E_{ikt}\,, k \ \in N, t \leq t_{max} \tag{28}$$

$$C_k + 1 - \sum_{i \in M} p_{ik} Y_{ik} \leq \sum_{i \in M} t E_{ikt} + M\left(1 - \sum_{i \in M} E_{ikt}\right), i \in M, k \ \in N, t \leq t_{max} \tag{29}$$

$$\sum_{t \leq t_{max}} F_{ijkt} \geq s_{ijk} X_{ijk}\,, i \in M, j \in N_0, k \ \in N, j \neq k \tag{30}$$

$$B_k \geq \sum_{i \in M, j \in N_0} t F_{ijkt}\,,, k \ \in N, t \leq t_{max} \tag{31}$$

$$B_k + 1 - \sum_{i \in M, j \in N_0} s_{ijk} X_{ijk} \leq \sum_{i \in M, j \in N_0} t F_{ijkt} + M\left(1 - \sum_{i \in M, j \in N_0} F_{ijkt}\right), k \in N, t \leq t_{max} \tag{32}$$

$$r_{max}^{P} \geq \sum_{i \in M, k \in N} r_{ik}^{P} E_{ikt}\,, t \leq t_{max} \tag{33}$$

$$r_{max}^{S} \geq \sum_{i \in M, j, k \in N, j \neq k} r_{ijk}^{S} F_{ijkt}\,, t \leq t_{max} \tag{34}$$

$$r_{max}^{H} \geq \sum_{i \in M, j, k \in N} r_{ik}^{HP} E_{ikt} + \sum_{i \in M, j, k \in N, j \neq k} r_{ijk}^{HS} F_{ijkt}, t \leq t_{max} \quad (35)$$

3. MÉTODOS E APLICAÇÃO

Este capítulo encontra-se dividido em 7 principais secções. No primeiro subcapítulo é aprofundado e detalhado o caso de estudo. Em 3.2. são destacados os principais contributos dos modelos adaptados. O subcapítulo 3.3. exibe os modelos desenvolvidos para responder ao problema do escalonamento de Máquinas Paralelas dedicadas com Setups dependentes da sequência de famílias de *Setups*, Recursos adicionais e datas de entrega (PMSR). O método para a resolução do problema com função multi-objetivo é explicada no subcapítulo 3.4. Em 3.5. explora-se as estratégias *de* heurísticas matemáticas utilizadas para apoiar o modelo na obtenção de melhores soluções. No subcapítulo 3.6. revela-se como foram geradas as instâncias que serão utilizadas para a obtenção de resultados. Por fim, em 3.7, expõe-se os parâmetros utilizados para o GRASP e o método de seleção.

3.1. Descrição do Problema

Este estudo tem como foco o departamento de produção da INPLAS, uma empresa que produz peças plásticas por injeção que serão utilizadas na indústria automóvel. Embora este estudo esteja centrado em um cenário específico, os modelos apresentados são aplicáveis para outras indústrias que compartilham as mesmas características.

Esta empresa enquadra-se num mercado exigente, sendo fundamental o escalonamento das tarefas para garantir o nível de serviço, permitindo responder a pedidos de peças plásticas num tempo de resposta de 48h. Para atender a essa procura, a produção é feita por turnos. O número de operadores é conhecido no início de cada turno, o que exige que o sistema seja rápido o suficiente para fornecer soluções no início do turno.

O estudo trata de um problema de escalonamento de máquinas paralelas dedicadas, onde *n* tarefas são processadas em *m* máquinas, sendo processadas em uma única máquina sem interrupção. Para processar cada tarefa é necessário um número específico de recursos de processamento. Os operadores são recursos de processamento, sendo apenas necessários enquanto as tarefas são processadas. Além disso, cada peça tem de ser produzida num molde específico. Embora um único molde possa produzir várias peças introduzindo diferentes composições de materiais, ele só pode ser usado em uma única máquina. A atribuição dos moldes às máquinas resulta de uma indicação técnica da empresa.

É impossível operar várias peças em simultâneo na mesma máquina. As trocas de molde exigem equipas especializadas e o número de equipa encontra-se limitado ao número de pontes rolantes existentes. As equipas de *setup* têm disponibilidade limitada e cada uma só pode efetuar uma troca de molde de cada vez.

Um conjunto de peças que usam o mesmo molde são consideradas da mesma família. Agrupar várias peças de uma mesma família é vantajoso porque elimina a necessidade de trocar o molde. Neste caso, o *setup* é dependente da sequência de famílias e realiza-se apenas entre trabalhos pertencentes a famílias diferentes.

Durante o processamento de uma tarefa não é permitido realizar uma troca de molde na mesma máquina, e vice-versa. Os recursos são dinâmicos pois a quantidade de recursos alocados para uma máquina flutua ao longo do tempo.

Este trabalho também contempla as condições iniciais das máquinas, crucial em processos produtivos contínuos. É fundamental identificar se há tarefas pendentes do dia anterior no início de um novo dia e, nesse caso, o tempo restante necessário para concluí-las será considerado como o tempo de processamento da tarefa inicial da máquina. Devido às condições variáveis entre os diferentes dias, não há garantia de que a máquina possa continuar a operar continuamente essas tarefas sem interrupções (podendo ter de parar a máquina no início do programa de produção), o que levaria a um tempo de conclusão da tarefa inicial superior o tempo de processamento da mesma. Mesmo na ausência de trabalhos iniciais, o conhecimento do último molde utilizado numa máquina pode ser fundamental para decidir o sequenciamento da máquina no dia seguinte.

O objetivo do departamento de produção é responder à procura diária de peças dentro dos limites de capacidade, datas de entrega, disponibilidade de operadores e equipas de *setup*.

3.2. Contribuições nas Adaptações dos Modelos Matemáticos

Este subtópico destaca os principais contributos das adaptações realizadas nos modelos de referência. Os contributos em questão são, nomeadamente: a adaptação do modelo geral de máquinas não relacionadas para o caso particular de máquinas dedicadas, incrementação da componente de datas de entrega, implementação de configurações inicias em cada máquina, a generalização do modelo para lidar com *setup* dependentes da sequência de famílias e a proposta de uma nova função multi-objetivo.

3.2.1. Caso particular das máquinas paralelas dedicadas

Os modelos adaptados e apresentados no capítulo 3.3. Modelos Matemáticos para PMRS resultaram da adaptação dos modelos desenvolvidos por (Fanjul-Peyro, 2020). O artigo mencionado utiliza esses modelos para responder ao problema de máquinas paralelas não relacionadas. Neste tipo de problema a alocação das tarefas às máquinas é um dos objetivos sendo possível que qualquer tarefa seja processada em qualquer máquina.

Se no modelo de referência o tempo de processamento de cada tarefa estava dependente da máquina onde esta era executada (p_{ij}), no caso das máquinas dedicadas o tempo de processamento não irá depender da máquina, visto que esta já está afeta (p_j). Da mesma forma que os tempos de processamento variavam com a máquina, também os tempos de *setup* tinham esta particularidade que se perde na adaptação para o caso particular ($s_{ijk} \rightarrow s_{jk}$).

Além disso, deixa também de ser necessário existir uma variável que sinalize a máquina em que a tarefa foi alocada (Y_{ik}). Desta forma a alocação às máquinas é previamente guardada nos *m* conjuntos N_i – conjuntos que apenas incluem as tarefas a serem processadas na máquina *i* – sendo necessário adaptar as restrições a este novo mecanismo de funcionamento. Neste sentido, em restrições como a (54), em que é necessário identificar o número de recursos a serem utilizados em simultâneo pelas diferentes máquinas, deve garantir-se que se comparam tarefas de máquinas diferentes, distinguindo por um índice a máquina de cada tarefa (N_i e N_q). Por outro lado, se o

objetivo for compreender o momento de tempo em que a tarefa pode ser executada na máquina, esta tarefa deve ser apenas comparada com outras tarefas da mesma máquina, como na restrição (43).

3.2.2. Resolução de problemas com datas de entrega

A implementação de datas de entrega é um dos aspetos inovadores face ao modelo de referência. Esta componente vem alterar aquilo que se deseja como objetivo que passará a ser a minimização dos atrasos.

O atraso de uma tarefa é considerado 0 se esta for concluída antes da data de entrega, no entanto, toma o valor da diferença entre a data de conclusão e a data de entrega ($C_j - d_j$) caso contrário. Seguindo este raciocínio, a função objetivo igual à equação (36):

$$Min\ Z = \sum_{j \in N} Max\ (C_j - d_j;\ 0) \tag{36}$$

Incorporar a fórmula do atraso na função objetivo associado iria introduzir não linearidades. Este aspeto tornaria o problema mais complexo.

Para que o problema tenha uma função objetivo linear, que geralmente permite algoritmos de otimização mais eficientes e podem simplificar a formulação do problema, criou-se a variável auxiliar T_j que representa o *tardiness* de cada tarefa. As equações (59) e (60) limitam o valor da variável, permitindo que T_j assuma o correto valor de *tardiness*. Em (37) é possível ver a componente da função objetivo relativa à minimização do atraso.

$$Min\ Z = \sum_{j \in N} T_j \tag{37}$$

3.2.3. Configuração inicial das máquinas

Tal como previamente abordado no capítulo 3.1. Descrição do Problema, conhecer a configuração inicial das máquinas – se existe alguma tarefa que iniciou o seu processamento no dia anterior e permanece no dia corrente e/ou qual o último molde que foi executado naquela máquina – é um dado importante para o escalonamento do dia corrente.

Esta contribuição é particularmente necessária quando procuramos resolver problemas de empresas reais com sistemas de produção contínuos.

Esta adaptação do modelo requer vários cuidados, nomeadamente:

- Garantir que a tarefa inicial da máquina (que transitou do dia anterior) será a primeira tarefa a ser executada naquela máquina no dia corrente. Por este motivo implementou-se as restrições (40).
- Compreender que a disponibilidade de recursos nos diferentes dias não é necessariamente igual e que esta atividade poderá ter de ser interrompida, o que fará com que o tempo de

conclusão da tarefa possa ser superior ao tempo de processamento. Surge assim as restrições (45).

Referir que a autora não encontrou na literatura qualquer modelo matemático que tivesse em consideração esta realidade da indústria.

3.2.4. Generalização do modelo para *setups dependentes da sequência de famílias*

O estudo do modelo de referência permitiu constatar que este não se encontra pronto para lidar com *setups* dependentes da sequência de famílias, isto é, não está preparado para a existência de tarefas que não requerem *setup*.

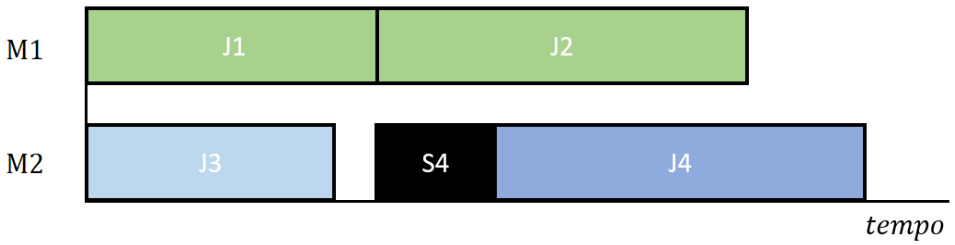


Figura 1 – Capacidade de lidar com inexistência de *setups* – pré adaptação

A Figura 1 é um exemplo elucidativo da imposição colocada pelas restrições do modelo de (Fanjul-Peyro, 2020). Consideremos que a disponibilidade dos recursos de *setup* é apenas 1 equipa de *setup*. Apesar de a equipa de *setup* se encontrar disponível imediatamente após a conclusão da tarefa 3 (J3), o modelo impede que o *setup* da tarefa 4 (S4) seja executado de seguida. Isto acontece porque o modelo considera que o momento em que termina o processamento da tarefa 1 (J1) e inicia do processamento da tarefa 2 (J2) é um *setup* e este não pode ser simultâneo à execução de um outro.

Desta forma, as restrições (47), comum aos dois modelos, resultaram de uma adaptação que consiste em apenas permitir que estas restrições sejam executadas, se e apenas se existir *setup* ($s_{jk} > 0$).

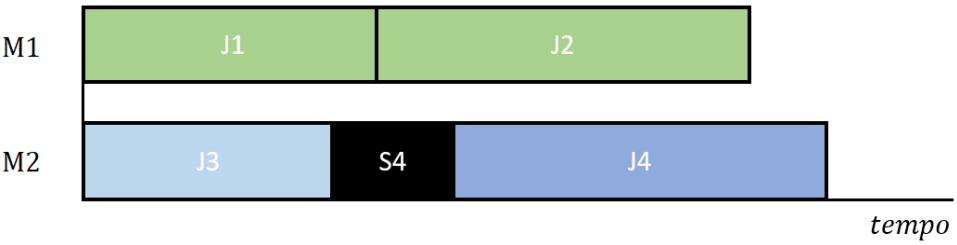


Figura 2 – Capacidade de lidar com inexistência de *setup*s - pós adaptação

Esta adaptação permite obter uma solução mais compacta e sem tempos ociosos desnecessários, Figura 2.

3.2.5. Função multi-objetivo

No modelo original, a função objetivo é a minimização do *makespan*, no entanto, visto que neste problema é importante procurar garantir o cumprimento das datas de entrega, obtendo a solução menos penalizadora em termos de *tardiness*, a função objetivo foi alterada para minimizar esta componente.

Esta função objetivo pode permitir muitas soluções ótimas e sobretudo, soluções ótimas muito diferentes umas das outras, podendo a solução final apresentada ser qualquer uma delas. Assim, em detrimento de soluções mais compactas, poderiam ser exibidas soluções com mais tempos ociosos entre as tarefas em cada máquina ou que promoviam mais trocas de *setup* que as necessárias. As Figuras 3 e 4 retratam soluções com o mesmo *tardiness* (de 1 unidade de tempo promovido pela tarefa J2), no entanto, são diferentes entre si no que toca à sequenciação de tarefas nas máquinas (o que promove um *setup* extra na tarefa 5 da Figura 3) bem como a existência de tempos ociosos não necessários. A Figura 4, por outro lado, tem uma representação muito mais compacta.

Neste sentido optou-se por tornar a função objetivo em multi-objetivo. Após a garantir a solução com menor *tardiness*, o modelo deve procurar a solução que minimize a soma dos *makespan* de todas as máquinas. Esta adaptação permite obter a solução da Figura 4.

Relativamente às Figuras 3 e 4, saliente-se que os retângulos coloridos simbolizam o processamento das tarefas identificadas por um código (p.e., J1, tarefa 1). Também os retângulos pretos, que simbolizam os *setups,* estão identificados por um código (p.e., S2, *setup* da tarefa J2). Além disso, outras informações individuais de cada tarefa, como o tempo de processamento de (p_k), a duração do tempo de *setup* (s_{jk}) e a data de entrega (d_k), estão assinaladas no respetivo retângulo. Importante mencionar que tarefas da mesma exata cor são tarefas da mesma família.

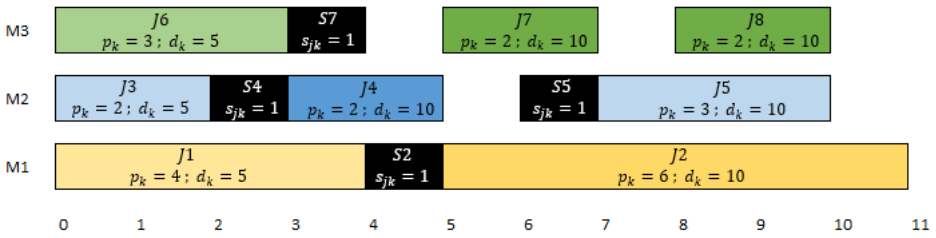


Figura 3 – Possível solução com função objetivo: minimização *tardiness*

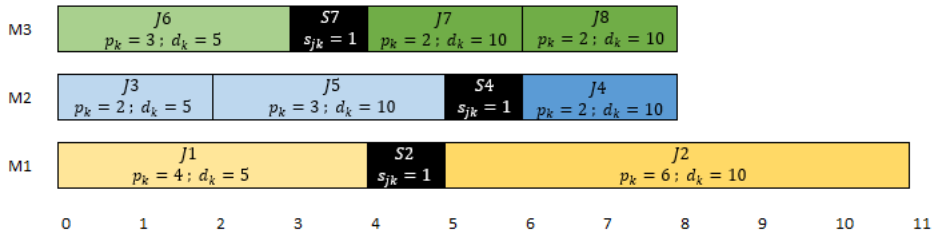


Figura 4 – Solução com função multi-objetivo

Optou-se pela minimização da *makespan* de cada máquina em detrimento da *makespan* total (função objetivo do modelo de referência), por se estar a trabalhar com máquinas dedicadas, e,

neste caso, a minimização do *makespan* afetar apenas a máquina gargalo, não evitando tempos ociosos nas demais máquinas. A função objetivo proposta fornece o cronograma de produção mais compacto possível que minimize o *tardiness*, permitindo a minimização simultânea dos tempos de paragem, do número de *setups* das máquinas e a maximização da utilização das máquinas.

Do conhecimento da autora, nenhuma pesquisa existente explorou a minimização do *tardiness* aliado à minimização da soma dos *makespan* de todas as máquinas como uma função objetivo, o que revela uma componente inovadora deste estudo.

A resolução deste problema ocorrerá pela programação lexicográfica por objetivos. Na programação lexicográfica, os vários critérios são ordenados pela sua importância. A estratégia consiste em encontrar a melhor solução para o critério principal, mantendo-a fixa, e, em seguida, otimizar o próximo critério sem prejudicar a solução obtida anteriormente. O processo é repetido para cada critério, afinando e melhorando a solução a cada passo, até que todos os critérios tenham sido considerados. Neste trabalho, o critério prioritário será a minimização do *tardiness*. Após obter o resultado do modelo para a minimização do *tardiness*, este valor irá limitar o campo de soluções para o segundo critério – minimização da soma dos *makespan* de todas as máquinas.

3.3. Modelos Matemáticos para o PMRS

O subcapítulo dedica-se particularmente à apresentação dos modelos matemáticos desenvolvidos para responder ao problema da programação de máquinas paralelas dedicadas com *setups* dependentes da sequência de famílias, com recursos adicionais e datas de entrega (PMRS). Inicialmente são explorados os parâmetros e variáveis a utilizar em ambos os modelos. Os subcapítulos seguintes mostram em detalhe cada um dos modelos, lógica subjacente aos mesmos e explicação das suas restrições.

3.3.1. Definição de parâmetros e variáveis

Nesta secção, define-se os parâmetros e variáveis comuns às duas abordagens de modelos para a resolução do problema. Para a definição formal de variáveis neste artigo, usamos letras minúsculas para se referir a parâmetros e letras maiúsculas para se referir a variáveis.

Os conjuntos e parâmetros usados são os seguintes:

- Um conjunto de *m* máquinas em que podem ser processados os trabalhos, denotada como $M = \{1, \dots, m\}$, indexado como *i*.
- Um conjunto de *m+n* trabalhos a ser processados, $N = \{1, \dots, m + n\}$, indexado por *j*, onde os primeiros *m* trabalhos representam os trabalhos iniciais da máquina com o respetivo número (*j* = 1 é a tarefa inicial da máquina 1, *j* = 2 é a tarefa inicial da máquina 2, …, *j* = *m* é a tarefa inicial da máquina *m*).
- p_j denota o tempo necessário para processar a tarefa *j* (minutos).
- s_{jk} denota o tempo necessário para fazer o *setup* entre as tarefas *j* e *k* (minutos).
- d_j denota a data de entrega da tarefa *j* (minutos).

- r_{max}^{P} é a quantidade máxima de recursos de processamento. Eles são necessários especificamente para trabalhos de processamento e não podem ser usados para trabalhos de *setup*. Denotamos esses recursos como P-recursos.
- r_{max}^{S} são a quantidade máxima de recursos de *setup*. Eles são necessários especificamente para a realização de *setup* e não podem ser usados para processamento de trabalhos. Denotamos esses recursos como S-recursos.
- r_j^P são os P-recursos necessários para fazer o processamento da tarefa *j*.
- r_j^S são S-recursos necessários para fazer o *setup* do trabalho *j*.
- *M* é uma constante suficientemente grande.

Como as máquinas são dedicadas, o conjunto de *N* trabalhos é dividido em *m* subconjuntos $(N_1, \dots, N_m)$ de modo que os trabalhos do conjunto N_i sejam processados apenas na máquina *i*. Cada máquina tem um trabalho fictício (0) representando seu início e fim. Assim, $N_{i0} = N_i \cup \{0\}$ representa o conjunto de trabalhos que serão processados na máquina *i* juntamente com o trabalho fictício. Além disso, os tempos de processamento e *setup* (p_j e s_{jk}), bem como os recursos necessários para cada operação (r_j^P e r_j^S) são fixos, pois cada tarefa é atribuída a priori a uma máquina.

Segue-se as variáveis utilizadas:

- $X_{ijk} = 1$ se a tarefa *k* for processada imediatamente após a tarefa *j* na máquina *i*, zero caso contrário.
- $Cmax_i$ é o tempo de conclusão da máquina *i* (minutos).
- C_j é o tempo de conclusão da tarefa *j* (coordenada onde o processamento da tarefa *j* termina na dimensão do tempo) (minutos).
- B_j é o tempo de conclusão do *setup* antes do processamento da tarefa *j* na correspondente máquina (coordenada onde a configuração do trabalho *j* termina na dimensão do tempo) (minutos).
- T_j é o *tardiness* da tarefa *j*, tomando o valor de $C_j - d_j$, na eventualidade deste ser positivo, ou 0, caso contrário (minutos).

3.3.2. Modelo *Strip-Packing*

Em (Fanjul-Peyro et al., 2017) são usados conceitos de *bin-packing* para criar um modelo matemático para o problema de UPMR usando a programação inteira mista. Em (Fanjul-Peyro, 2020) esta abordagem é adaptada para responder ao problema de UPMSR. *Bin-packing* envolve a organização de *itens* retangulares dentro de uma caixa retangular, sendo que as tarefas são representadas por *itens* e as máquinas por *bins*. Na variante 2D, pretende-se encaixar itens retangulares em uma caixa de largura fixa, minimizando a outra dimensão. Cada retângulo tem uma altura e uma largura. Para aproximar o problema à realidade do PMRS podemos considerar cada tarefa como um retângulo com largura determinada pelo seu tempo de processamento (p_j) e altura determinada pelos recursos necessários (r_j^P). No *Strip-Packing*, precisamos saber a posição

de cada retângulo, que é representado pelas coordenadas de seu canto superior direito. O exemplo apresentado (Figuras 5 e 6) considera que o eixo *x* representa o tempo de processamento e o eixo *y* representa o número de operadores (recursos de processamento). Portanto, as coordenadas são denotadas como C_j para tempo e R_j^P para recursos.

Nas Figuras 5 e 6 existem 3 tarefas que necessitam ser escalonadas. A tarefa J1 está afeta à máquina 1 e as tarefas J2 e J3 estão afetas à máquina 2. A tarefa J1 é previamente alocada. Na máquina 2, existe duas opções de sequenciamento: J2-J3 ou J3-J2. A opção de sequenciamento J2-J3 não permite que a tarefa J2 ocorra em simultâneo com a tarefa J1 uma vez que extrapolaria o número de recursos máximos disponíveis. Consequentemente, o início do processamento da tarefa J2 só iniciaria após o término da tarefa J1. A tarefa J3 apenas poderá iniciar quando a tarefa da mesma máquina (J2) terminar.

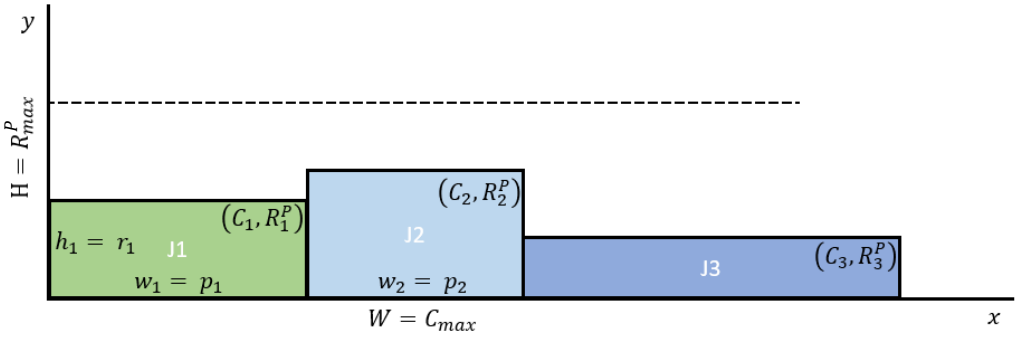


Figura 5 – *Strip-Packing* para PMRS (a)

Por outro lado, a sequência J3-J2 permite que a tarefa J3 ocorra em simultâneo com a tarefa J1 uma vez que permite o cumprimento da restrição de recursos. A tarefa J2 apenas poderá iniciar quando a tarefa da mesma máquina (J3) terminar.

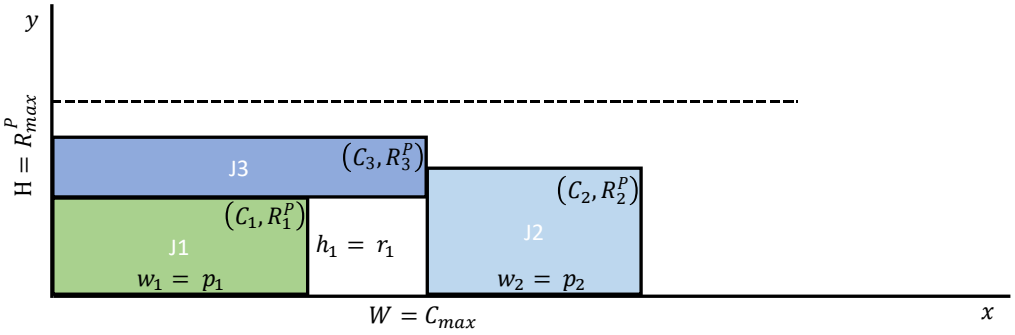


Figura 6 – *Strip-Packing* para PMRS (b)

Salientar que como existe dois tipos de recurso, este raciocínio aplica-se também aos recursos de *setup*. Nesse caso a largura dos retângulos é determinada pelo seu tempo de *setup* (s_{jk}) e altura pelas equipas de *setup* necessários (r_j^S), sendo que as coordenadas de cada ponto serão compostas por (B_j, R_j^S).

Este problema carece da adição de outras variáveis. As letras em sobrescrito são usadas para indicar a dimensão em que são usadas.

- $R_j^P \in [0, r_{max}^P]$ é a coordenada de processamento da tarefa *j* na dimensão P-recursos (os recursos específicos empregados no processamento da tarefa *j*).
- $R_j^S \in [0, r_{max}^S]$ é a coordenada de configuração da tarefa *j* na dimensão S-recursos (os recursos específicos empregados na configuração antes do processamento da tarefa *j*).
- D_{jk}^P, E_{jk}^P, F_{jk}^S, G_{jk}^S são variáveis binárias auxiliares com valor = 1 quando o valor da variável estudada nas restrições (tempo ou recursos) da tarefa *k* é maior que o valor da variável estudada (tempo ou recursos) da tarefa *j*.

$$Min\ Z1 = \sum_{j \in N} T_j \tag{38}$$

$$Min\ Z2 = \sum_{i \in M} Cmax_i \tag{39}$$

$$X_{i0i} = 1, i \in M \tag{40}$$

$$\sum_{j \in N_i, j \neq k} X_{ijk} = 1, i \in M, k \in N_i \backslash \{i\} \tag{41}$$

$$\sum_{j \in N_{i0} \backslash \{i\}, j \neq k} X_{ikj} = 1, i \in M, k \in N_i \tag{42}$$

$$C_k - C_j + M\left(1 - X_{ijk}\right) \geq s_{jk} + p_k, i \in M,\ j \in N_i, k \in N_i \backslash \{i\}, j \neq k \tag{43}$$

$$C_k \geq \sum_{j \in N_{i0}, j \neq k, s_{jk} > 0} X_{ijk}\, s_{jk} + p_k, k \in N_i \backslash \{i\}, i \in M \tag{44}$$

$$C_i \geq p_i, i \in M \tag{45}$$

$$Cmax_i \geq C_k, i \in M, k \in N_i \tag{46}$$

$$B_k - C_j + M\left(1 - X_{ijk}\right) \geq s_{jk},\ i \in M, j \in N_i, k \in N_i \backslash \{i\}, j \neq k, s_{jk} > 0 \tag{47}$$

$$B_k \leq C_k - p_k, k \in N \tag{48}$$

$$r_{max}^P \geq R_k^P, k \in N \tag{49}$$

$$R_k^P \geq r_k^p, k \in N \tag{50}$$

$$r_{max}^S \geq R_k^S, k \in N \tag{51}$$

$$R_k^S \geq r_k^S, k \in N \tag{52}$$

$$C_k - C_j + M\left(1 - D_{jk}^P\right) \geq p_k, i, q \in M, q \neq i, j \in N_i, k \in N_q \tag{53}$$

$$R_k^P - R_j^P + M\left(1 - E_{jk}^P\right) \geq r_k^P, i, q \in M, q \neq i, j \in N_i, k \in N_q \tag{54}$$

$$D_{jk}^P + D_{kj}^P + E_{jk}^P + E_{kj}^P \geq 1, i, q \in M, q \neq i, j \in N_i, k \in N_q, j \neq k \tag{55}$$

$$B_k - B_j + M\left(1 - F_{jk}^S\right) \geq \sum_{l \in N_{q0}, l \neq k, s_{lk} > 0} X_{qlk}\, s_{lk}, i, q \in M, q \neq i,\ j \in N_i, k\ \in N_q \quad (56)$$

$$R_k^S - R_j^S + M\left(1 - G_{jk}^S\right) \geq r_k^S, i, q\ \in M, q \neq i,\ j \in N_i, k\ \in N_q \quad (57)$$

$$F_{jk}^S + F_{kj}^S + G_{jk}^S + G_{kj}^S \geq 1, i, q\ \in M, q \neq i\ , j \in N_i, k\ \in N_q \quad (58)$$

$$T_j \geq C_j - \ d_j, j \in N \quad (59)$$

$$X_{ijk} \in \{0,1\}, T_k \geq 0, C_k \geq 0, B_k \geq 0\ j, k \in N_i, i \in M \quad (60)$$

Este é um problema multi-objetivo. O primeiro objetivo (38) procura minimização do *tardiness*, isto é, procura-se a solução que provoque menos atrasos face às datas de entrega. A minimização da soma dos *makespan* de todas as máquinas (39) é o segundo objetivo deste problema.

As restrições (40) garantem que as tarefas iniciais sejam as primeiras a serem processadas imediatamente após as tarefas fictícias em cada máquina. As restrições (41) e (42) garantem que apenas uma tarefa seja processada imediatamente antes e apenas uma tarefa seja processada imediatamente após cada tarefa (condições de fluxo). As restrições (43) quebram sub-rotas e fornecem a ordem correta de escalonamento, obrigando a que, se *k* for processado depois de *j* na máquina *i*, a conclusão do processamento da tarefa *k* não aconteça antes de s_{jk}+p_k. As restrições (44) definem o limite inferior do tempo de conclusão da tarefa *k* (C_k), forçando-a a ser maior ou igual à soma do seu tempo de *setup* com o seu tempo de processamento, enquanto as restrições (45) fazem o mesmo para as tarefas iniciais. As restrições (46) estabelecem a relação entre $Cmax_i$ da máquina *i* e os tempos de conclusão das tarefas naquela máquina. As restrições (47) impõem que, se a tarefa *j* for a tarefa anterior à tarefa *k* na máquina *i*, o tempo de término do *setup k* deve ser posterior ao tempo de término do processamento da tarefa *j* somado ao tempo de *setup* da tarefa *k*. As restrições (48) garantem que a execução do *setup* do trabalho *k* ocorre antes processamento do mesmo trabalho *k*. As restrições (49)-(52) definem os limites de coordenadas de recursos específicos usados no processamento $\left(R_k^P\right)$ e usados especificamente nas configurações $\left(R_k^S\right)$. As restrições (53) mostram que, se *k* for processado depois de *j*, a diferença entre esses tempos de conclusão $(C_k - C_j)$ é maior ou igual ao tempo de processamento da tarefa *k*. Em (54) se R_k^P for maior que R_j^P, a diferença entre as coordenadas desses pontos de recursos é maior ou igual aos recursos utilizados no processamento do trabalho *k*. As restrições (55) forçam pelo menos uma das variáveis binárias a ter o valor 1. As restrições (56) mostram que, se *k* ocorrer posteriormente a *j*, a diferença entre os seus tempos de finalização do *setup* $(B_k - B_j)$ é maior ou igual ao tempo de *setup* de trabalho *k*. Em (57) se R_k^S for maior que R_j^S, a diferença entre as coordenadas desses pontos de recursos é maior ou igual aos recursos utilizados na configuração do trabalho *k*. As restrições (58) forçam pelo menos uma das variáveis binárias a ter o valor 1. As restrições (59) permitem a contabilização do *tardiness* (tempo associado ao atraso) de cada tarefa, tomando o valor de 0 se a data de entrega (d_j) for superior à conclusão da tarefa (C_j) ou, caso contrário, o valor de $C_j - \ d_j$. Por fim, as restrições (60) definem os limites das variáveis. Neste modelo, além de várias simplificações do modelo original, as funções objetivo (38)-(39) e as restrições (40) e (59) são novas e foram introduzidas para refletir as condições específicas do problema real.

3.3.3. Modelo *Time Index*

Modelo *Time Index* é um modelo matemático onde as variáveis são indexadas individualmente para cada unidade de tempo, tratando o tempo como uma dimensão discreta. Este tipo de modelo permite a análise e otimização de variáveis e restrições em um horizonte de tempo definido, discriminando os aspetos específicos em cada instante.

Este problema carece da adição de outras variáveis/parâmetros. E_{ikt} e F_{ijkt} são variáveis de decisão e t_{max} é um parâmetro.

- $E_{ikt} = 1$ se *k* estiver a ser processado na máquina *i* no tempo *t*, 0 caso contrário.
- $F_{ijkt} = 1$ se existir *setup* entre a tarefa *j* e a tarefa *k*, na máquina *i*, no tempo *t*, 0 caso contrário.
- t_{max} é o horizonte temporal do planeamento.

Para este MILP (Modelo de Programação Linear Inteira), usar-se-ão as equações (38) – (48) e (59) – (60), comuns ao *Strip-Packing*. As restrições (61)-(68) são específicas para o modelo *Time Index*.

$$\sum_{t \le t_{max}} E_{ikt} \ge p_k \,, i \in M, k \ \in N_i \tag{61}$$

$$C_k \ge tE_{ikt}, i \in M, k \ \in N_i, t \le t_{max} \tag{62}$$

$$C_k + 1 - p_k \le tE_{ikt} + M(1 - E_{ikt}), i \in M, k \ \in N_i, t \le t_{max} \tag{63}$$

$$\sum_{t \le t_{max}} F_{ijkt} \ge s_{jk} X_{ijk} \,, i \in M, j \in N_i, k \ \in N_i, j \ne k, s_{jk} > 0, k \ge m \tag{64}$$

$$B_k \ge \sum_{j \in N_i, j \ne k} tF_{ijkt} \,, i \in M, k \ \in N_i, t \le t_{max}, s_{jk} > 0, k \ge m \tag{65}$$

$$B_k + 1 - \sum_{j \in N_i, j \ne k, s_{jk} > 0} s_{jk} X_{ijk} \le \sum_{j \in N_i, j \ne k, s_{jk} > 0} tF_{ijkt} + M\left(1 - \sum_{j \in N_i, j \ne k, s_{jk} > 0} F_{ijkt}\right), i \in M, k \in N_i, t \le t_{max} \tag{66}$$

$$R^P_{max} \ge \sum_{i \in M, k \in N_i} r^P_k E_{ikt} \,, t \le t_{max} \tag{67}$$

$$R^S_{max} \ge \sum_{i \in M, j, k \in N_i, j \ne k, s_{jk} > 0} r^S_k F_{ijkt} \,, t \le t_{max} \tag{68}$$

As restrições (61) e (62) trabalham juntas para garantir que a variável E_{ikt} tome o valor de 1 apenas durante os instantes *t* em que a tarefa *k* está sendo processada na máquina *i*. Isso evita tempo ocioso desnecessário na máquina e garante que cada tarefa seja processada pelo tempo mínimo necessário para concluí-la. As restrições (63) complementam essas restrições exigindo que quando E_{ikt} for igual a 1, o tempo atual *t* deve ser maior ou igual ao tempo de conclusão da tarefa *k* menos seu tempo de processamento mais 1. As restrições (64)-(66) representam o mesmo para *setups*. As

restrições (67) garantem que a soma de todos os recursos utilizados no processamento em cada instante *t* seja menor ou igual ao número máximo de recursos disponíveis para processamento. A restrição (68) garante o mesmo para *setups.*

3.4. Método das Duas Fases

O problema apresentado carece de uma abordagem multiobjectivo. Uma vez que, em primeiro lugar, se deve minimizar o *tardiness*, este será o principal objetivo. Após obter o melhor *tardiness*, o modelo deve procurar a solução que minimize a soma dos *makespan* de todas as máquinas. Para resolver este problema aplicou-se o método das duas fases – fluxograma da Figura 7.

Este método consiste em duas fases: resolução do modelo exato apenas para com a minimização do principal objetivo (38) e resolução do modelo exato apenas com a minimização do segundo objetivo (39). Note-se que, de forma a respeitar o valor obtido pela minimização do *tardiness* (1ª fase), a resolução do segundo modelo terá uma restrição que limita o valor de *tardiness* da nova solução (69).

$$\sum_{j \in N} T_j \leq Z1 \tag{69}$$

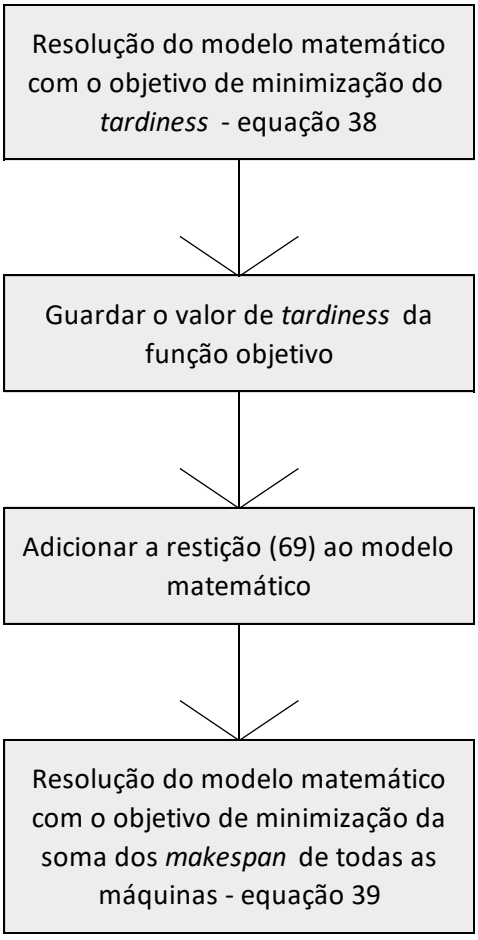


Figura 7 – Fluxograma do Método das Duas Fases

3.5. Heurísticas Matemáticas

Os modelos matemáticos ao resolver instâncias de maior dimensão tendem a ter um desempenho menos positivo uma vez que carece de um maior número de variáveis e restrições.

Heurísticas matemáticas é uma abordagem que combina a utilização de modelos matemáticos com heurísticas e/ou metaheurísticas. Algumas das estratégias inerentes a este conceito e desenvolvidas neste projeto são o *warm-start* que através de uma metaheurística permite fornecer solução inicial válida (limitando o campo de soluções superiormente – *upper bound*) e uma heurística construtiva que permite limitar inferiormente o campo de soluções (*lower bound*).

3.5.1. GRASP - *Greedy Randomized Adaptive Search Procedures*

O algoritmo GRASP (*Greedy Randomized Adaptive Search Procedure*) foi inicialmente desenvolvido por (Feo & Resende, 1989) e tem sido amplamente utilizado para resolver problemas combinatórios reais. Um dos campos de aplicação do GRASP é o problema de escalonamento.

Em (Feo et al., 1991) os autores propuseram um algoritmo GRASP para o problema de programação em máquina única, procurando minimizar o tempo de fluxo e o *earliness* (antecipação). Em (Feo et al., 1996) utilizaram um algoritmo GRASP para resolver o escalonamento em máquina única considerando tempos de *setup* dependentes da sequência e objetivando minimizar o *tardiness*. No contexto do problema de escalonamento em um *job shop*, (Aiex et al., 2003) e (Binato et al., 2002), desenvolveram algoritmos GRASP específicos. Essas abordagens demonstraram ser eficientes na obtenção de soluções próximas ao ótimo para problemas complexos em ambientes de *job shop*. Em (Rajkumar et al., 2011) foi apresentado um algoritmo GRASP para resolver o problema de agendamento flexível de oficinas de trabalho com restrições de recursos limitados, mostrando-se capaz de lidar com as restrições impostas e obter soluções satisfatórias para o problema. Recentemente, esta foi também a metaheurística utilizada por (Yepes-Borrero et al., 2020) para responder ao problema de máquinas não relacionadas com recursos adicionais.

Neste trabalho em particular, usar-se-á esta estratégia para encontrar uma boa solução completa. Concretamente, propõe-se um algoritmo GRASP que se reparte em 2 fases: fase construtiva (criação de uma solução com base no custo de cada tarefas) e fase *Local Search* (melhoria da solução obtida na fase construtiva através da aplicação de *swaps* internos), procedimento evidenciado na Figura 8.

A fase construtiva será executada considerando recursos. Esta abordagem foi selecionada em alternativa à construção de uma solução inicial e posterior aplicação de mecanismos de reparação uma vez que (Yepes-Borrero et al., 2020), para um problema semelhante, concluiu que seria esta a abordagem mais promissora.

```
procedure GRASP
    Read_Input():
    for k = 1, … , Constructions_Max_Iterations do
        Solution ← Greedy_Randomized_Construction;
        Update_Solution(Solution, Best_Solution);
        Solution ← Local_Search(Solution, Best_Solution);
        Update_Solution(Solution, Best_Solution);
    end;
    return Best_solution
end GRASP.
```

Figura 8 – Pseudocódigo da metaheurística GRASP

A aleatorização desempenha um papel fundamental na fase construtiva da otimização combinatória pois é utilizada para evitar ótimos locais. Inicialmente deve-se criar uma lista ordenada por critérios (CL). CL é ordenado de acordo com o custo das tarefas. Ao realizar o processo de atribuição, em vez de selecionar o melhor candidato de CL, opta-se por selecionar aleatoriamente da lista restrita de candidatos (RCL). O tamanho de RCL é determinado por um valor α ($\alpha \in [0,100]$), possuindo $\alpha\%$ primeiros dos elementos de CL - quanto mais próximo α estiver de 100%, maior será o tamanho da RCL. À semelhança de (Yepes-Borrero et al., 2020) que menciona ter utilizado métodos *multi-start*, propõe-se que a fase construtiva seja iterativa, estudando a vizinhança de diferentes soluções. A Figura 9 representa a fase descrita.

Comparativamente com (Yepes-Borrero et al., 2020), como estamos a tratar de um problema diferente, lidando com máquinas dedicadas, famílias de *setup* e objetivando-se uma solução que minimize os atrasos, optou-se por modificar os critérios de ordenação dos candidatos (CL), selecionado, em primeiro lugar, a tarefa com maior *tardiness*. Em caso de empate, optar pela tarefa que requer menor tempo de *setup* ou, em última instância, a tarefa que possuir menor *earliness.*

```
procedure Greedy_Randomized_Construction
    Solution ← ∅
    Evaluate the incremental costs of the candidate elements (tardiness; setup time; earliness)
    while Solution is not a complete solution do
        Create a candidate list (CL) according to the order criteria: tardiness; setup time; earliness
        Build the restricted candidate list (RCL) and Select a job j from the RCL at random;
        Solution ← Construction_Phase(Solution, j);
        Reevaluate the incremental costs
    end;
    return Solution
end Greedy_Randomized_Construction.
```

Figura 9 – Pseudocódigo da Fase *Greedy* do GRASP

A Figura 10 demonstra como se introduz a tarefa selecionada (*j*) na solução construída até ao momento. Apenas se apresenta a fase de construção do *setup* uma vez que a fase de construção do processamento segue a mesma linha de raciocínio. Em primeiro lugar, quando se procura introduzir uma tarefa na máquina, deve identificar-se quem será gargalo: a máquina ou os recursos necessários. Caso os recursos necessários já estejam disponíveis quando a máquina também fica disponível, a tarefa deve ser executada assim que a máquina terminar a tarefa anterior. Na situação oposta, a operação associada à tarefa só poderá começar quando todos os recursos necessários para o início da operação estiverem livres.

procedure Construction_Phase *(Solution, j)*

#Setup Construction Phase

setup[j] = s_{lj} #*l* is the last task processed in same machine as *j* will be processed

if *setup[j]* > 0 **then**:

for p = 1, … , R^{S}_{max} **do**

Identify the $\mathbf{r}_{\mathbf{j}}^{\mathbf{S}}$ that are the first setup resources to become available.

end;

highest_rsetup ← Later availability of the $\mathbf{r}_{j}^{\mathbf{S}}$ setup resources that become available first

use_machine [i] ← Current machine *i* makespan

if *use_machine[i]* ≥ *highest_rsetup* **then**:

The setup will start as soon as the machine becomes available;

From the setup resources available until the machine becomes accessible (*use_machine[i]*), choose the $\mathbf{r}_{j}^{\mathbf{S}}$ resources that become available last.

Update *Solution* (update: B_j, use_machine[i], availability of resources)

else:

The process will start as soon as the necessary amount of setup resources becomes available;

From the setup resources available until the *highest_rsetup*, choose the $\mathbf{r}_{\mathbf{j}}^{\mathbf{S}}$ resources that become available last.

Update *Solution* (update: B_j, use_machine[i], availability of resources)

#Process Construction Phase - Equivalent to what was demonstrated for the setups.

return *Solution*

end Construction_Phase

Figura 10 – Pseudocódigo da Fase Construtiva do GRASP

A fase de *Local Search* tem o objetivo de melhorar a solução obtida na fase construtiva. Todas as alterações executadas nesta fase carecem de validação e, caso se verifique incongruências, reparação. No algoritmo de referência são utilizadas 3 diferentes técnicas para a operação de *Local Search*: *swap* interno, *swap* externo e inserção externa, no entanto as últimas duas alternativas consistem na troca de tarefas entre máquinas, o que não é possível no problema estudado. Assim,

a operação a executar na fase do *Local Search* será o *swap* interno - para cada tarefa *j* em cada máquina *i*, testa-se uma troca entre a tarefa *j* e qualquer outra tarefa *k* processada na mesma máquina. A solução proveniente da fase *Greedy_Randomized_Construction* será atribuída à solução corrente (*Current_Solution*), sendo que esta sofrerá um *swap* interno para que seja criada uma solução de teste (*Test_Solution*). Uma vez que a ordem das tarefas se altera e os *setup*s são dependentes da sequência de famílias, deve recalcular-se os tempos de *setup*s antes de avançar para a fase de reparação. Obtendo uma solução de teste válida após a reparação, deve-se comparar o seu resultado com a solução corrente e a melhor solução do momento (*Best_Solution*). Sempre que for encontrado um melhor vizinho, assume-se essa solução como *solução corrente*, reiniciado o ciclo. Se a solução teste for melhor que a melhor solução atual, esta também deve ser substituída. O *Local Search* será aplicado à solução proveniente da fase construtiva enquanto não se atingir o critério de paragem: terminando quando não existir nenhuma solução vizinha melhor que a corrente (solução corrente ser um ótimo local) ou no fim da análise de *LS_Max_Iterations* soluções currentes e respetivos vizinhos. A Figura 11 ilustra o pseudocódigo associado a esta função.

```
procedure Local_Search(Solution, Best_Solution)
        Current_Solution ← Solution
        Test_Solution ← Solution
        while iteration < LS_Max_Iterations do
                Apply Internal_Swap(Current_Solution)
                The task setup times are recalculated.
                Test_Solution ← Repairing_Mechanism(Test_Solution)
                if Test_Solution < Best_Solution then
                        Best_Solution ← Test_Solution
                if Test_Solution < Current_Solution then
                        Current_Solution ← Test_Solution
                        iIteration = iteration + 1
                else
                        Test_Solution ← Current_Solution
                if Test_Solution verified the last possible change without improvement then
                         iteration = LS_Max_Iterations
        end;
        return Best_Solution
end Local_Search.
```

Figura 11 – Pseudocódigo da Fase Local Search

As trocas executadas na fase de *Local Search*, mais concretamente no *swap* interno podem levar a soluções não exequíveis. Por esse motivo, é necessário compreender se existe algum(alguns) período(s) em que o número de recursos seja excedido e, caso se verifique, arrastar a tarefa que termina mais tarde para que esta inicie no fim da tarefa que termina mais cedo. Aplicando-se mecanismo de reparação num dado instante, a iteração seguinte deve confirmar se esse mesmo momento ficou totalmente corrigido e, em caso afirmativo, avançar para o próximo instante. O exemplo das Figuras 12 e 13 permitem compreender o funcionamento do mecanismo de reparação para quando os recursos de processamento são ultrapassados, sendo que a lógica inerente aos recursos de *setup* é a mesma. Assim, pressupondo que o número de recursos de processamento disponíveis são 6, as tarefas J1 e J3 não poderão ser executadas simultaneamente (3 + 4 > 6). Assim, mover-se-á a tarefa que termina mais tarde (J3) para que esta inicie quando a tarefa J1 termina.

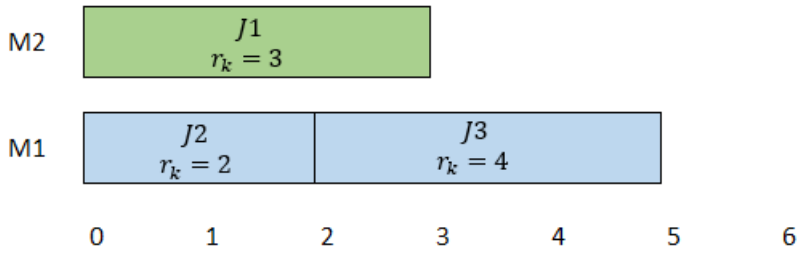


Figura 12 – Mecanismo de Reparação - pré reparação

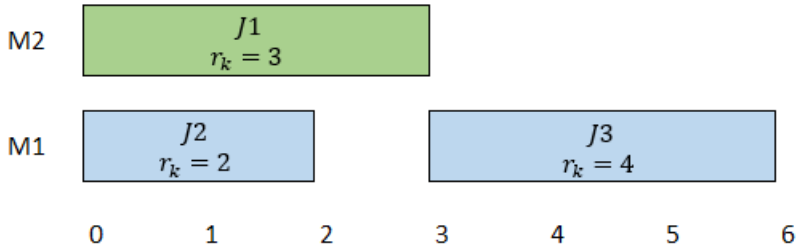


Figura 13 – Mecanismo de Reparação - pós reparação

O raciocínio inerente ao Mecanismo de Reparação encontra-se na Figura 14.

O mecanismo de reparação proposto por (Yepes-Borrero et al., 2020) pretende responder a um problema em que os únicos recursos existentes são de *setup.* Neste sentido, um dos contributos desta metaheurística é a extensão do mecanismo de reparação para que este também seja capaz de reparar soluções com recursos de processamento. Foi também necessário incorporar uma lógica para saber se, numa iteração, deveriam ser procuradas correções em recursos de *setup* ou em recursos de processamento. A lógica selecionada passa por identificar o momento mais cedo não analisado de processamento (*timetoanaliseP*) e o momento mais cedo não analisado de *setup* (*timetoanaliseS*), selecionando o inferior.

procedure Repairing_Mechanism(*Test_Solution*)

timetoanaliseS ← Shortest $\boldsymbol{B_j}$

timetoanaliseP ← Shortest $\boldsymbol{C_j}$

while *timetoanaliseS* ≤ $\boldsymbol{max}\left(\boldsymbol{B_j}\right)$ and *timetoanaliseP* ≤ $\boldsymbol{max}\left(\boldsymbol{C_j}\right)$ **do**:

if *timetoanaliseP* ≤ *timetoanaliseS* **then**: #Analise Process

if any moment required operators > $\mathbf{r}_{\boldsymbol{max}}^{\boldsymbol{P}}$ **then**:

Postpone the begining of the process in the machine that begins the latest out of those thar overlap at instant *t*.

else:

Update *timetoanaliseP.*

end

else: #Analise Setup

if any moment required *setup* teams > $\mathbf{r}_{\boldsymbol{max}}^{\boldsymbol{S}}$ **then**:

Postpone the beginning of the setup in the machine that begins the latest out of those thar overlap at instant t;

else:

Update *timetoanaliseS.*

end

end

end

end Repairing_Mechanism.

Figura 14 – Pseudocódigo do Mecanismo de Reparação do GRASP

3.5.2. Heurística RULS – *Lower bound*

Lower bound, ou limite inferior, representa um valor dado a uma determinada variável (ou a um conjunto delas) que atua como restrição. Neste caso em particular, é fornecido um *lower bound* à soma dos *makespan* de todas as máquinas – variáveis da 2ª função objetivo do modelo, equação (39) – limitando o campo de pesquisa e assim permitindo uma maior eficiência do modelo uma vez que este apenas procurará soluções com valor de soma dos *makespan* de todas as máquinas igual ou superior ao do *lower bound*.

"RULS" – "*Resources-Unlimited & the Least Setup times*" significa "recursos ilimitados e tempos de *setup* mínimos", que capta os elementos-chave da heurística, ou seja, define a *makespan* de cada máquina considerando recursos ilimitados e o menor tempo possível em operações de *setup*. Em cada máquina, examinamos o número máximo de famílias de *setup* existentes e supomos que, na melhor situação, o número de trocas de *setup* é igual ao número máximo de famílias menos um. Com base nesse valor, tentamos determinar os menores tempos de *setup* possíveis. Esses tempos são então somados ao tempo total de processamento de todas as tarefas executadas na máquina.

O *lower bound* é calculado somando esses resultados para todas as máquinas. A Figura 15 traduz o raciocínio descrito sob a forma de pseudocódigo.

```
procedure RULS_Heuristic
        Read_Input():
        for m = 1, … , Number_of_Machines do
                Find number of different setup families
                f = number of different setup families
                ms = minimum number of setups
                ms = f – 1
                Find the msº smallest setups possible in machine i
                Lowerbound[i] = msº smallest setups possible in machine i + sum of all processing times
                of machine i
        end;
        Total_Lowerbond = Sum (Lowerbound)
end RULS_Heuristic.
```

Figura 15 – Pseudocódigo da Heurística que fornece o *Lower bound*

Este *lower bound* será incorporado pelas restrições (70).

$$\sum_{i \in M} Cmax_i \geq Total\_Lowerbound \tag{70}$$

Demonstração

O exemplo seguinte contém 3 máquinas com 10 tarefas a serem processadas. Previamente foram alocadas as tarefas J1, J2 e J3 à Máquina 1, J4, J5, J6 e J7 à Máquina 2 e J8, J9 e J10 à Máquina 3.

A Tabela 3 associa cada tarefa com a sua família. A Tabela 4 mostra os tempos de *setup* necessários de realizar se a tarefa *k* da família *K* for realizada depois da tarefa *j da* família *J*.

Tabela 3 – Família de cada tarefa a ser processada

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Família	1	2	2	3	4	5	5	6	7	8

Tabela 4 – Tempo de *setup* entre tarefas da família J e da família K

s_{JK}	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	2	-	-	-	-	-	-
2	3	0	-	-	-	-	-	-
3	-	-	0	5	2	-	-	-
4	-	-	4	0	1	-	-	-
5	-	-	3	5	0	-	-	-
6	-	-	-	-	-	0	3	2
7	-	-	-	-	-	3	0	1
8	-	-	-	-	-	5	1	0

Caso não fosse necessário executar *setups* e os recursos de processamento fossem ilimitados, a *makespan* de cada máquina seria a soma dos tempos de processamento das tarefas a serem executadas na mesma. A Figura 16 representa o esquema de solução para este cenário.

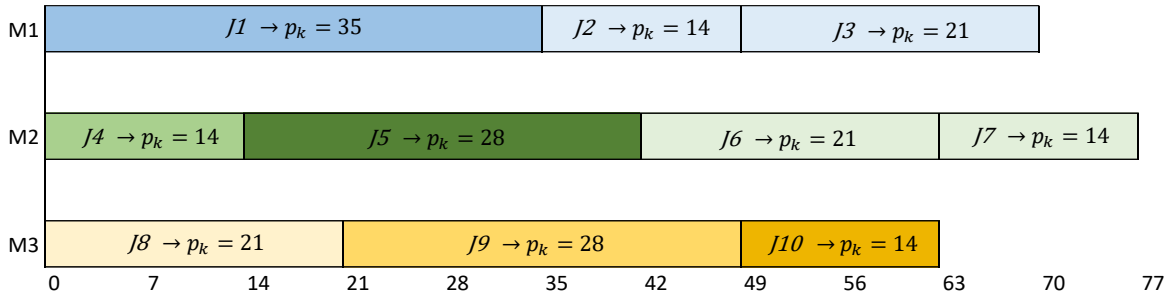


Figura 16 - Solução com recursos de processamento ilimitados e sem considerar *setups*

A soma dos *makespan* de todas as máquinas da solução da Figura 16 é um limite inferior (*lower bound*) da solução. No entanto, é possível tornar o *lower bound* ainda mais forte.

Sabendo que cada tarefa da mesma família está representada pela mesma cor (informação complementar à Tabela 3), a análise da Figura 16 permite identificar 2 famílias diferentes na M1, 3 famílias diferentes em M2 e M3. Assim é seguro afirmar que existe, pelo menos, 1 troca de molde (operação de *setup*) em M1 e 2 trocas de molde (operações de *setup*) em M2 e M3.

Recorrendo à Tabela 4, compreendemos que:

- Em M1, o menor tempo de *setup* possível é 2 min.
- Em M2, os 2 menores tempos de *setup* possíveis são: 1min e 2 min.
- Em M3, os 2 menores tempos de *setup* possíveis são: 1min e 1 min.

Deste modo podemos garantir que a *makespan* da M1 nunca será maior do que 70 minutos (*makespan* da M1 na Figura 16) mais 2 minutos de *setup*, ou seja, 72 minutos. O mesmo raciocínio aplica-se às restantes máquinas.

3.6. Geração de Instâncias

A geração de instâncias foi realizada tendo por base os dados reais da empresa. Assim, foram definidos o número de trabalhos, de máquinas, de equipas de *setup* e de operadores disponíveis bem como o período temporal para as datas de entrega. Os conjuntos de parâmetros para os problemas, pequenos, médios, grandes e extra grandes, são apresentados na Tabela 5.

Para cada um dos conjuntos, 8 instâncias foram geradas aleatoriamente os tempos de processamento, os tempos de *setup* e a famílias a que cada peça pertence a partir das distribuições de U(120,660), U(30,50) e U(1,3), respetivamente. Além disso, também o número de operadores necessário ao processamento dos trabalhos nas máquinas e o dia de entrega da tarefa foram gerados aleatoriamente segundo as respetivas distribuições U(1,4) e U(1,3). Estabeleceu-se também que cada operação de *setup* precisa apenas de 1 equipa. Os tempos de *setup* devem satisfazer a desigualdade triangular. Por fim, referir que a atribuição de tarefas às máquinas é um processo aleatório.

À semelhança de (Fanjul-Peyro, 2020), o número de operadores (recursos utilizados no processamento) disponíveis foi definido pelo resultado da equação (71), sendo que R_M^P representa o número de recursos máximos possíveis por tarefa e R_m^P o número mínimo de recursos de processamento mínimos por tarefa, que, respetivamente, correspondem a 4 e 1. *m* é o número de máquinas do problema.

$$\frac{R_M^P + R_m^P}{2} \times m \tag{71}$$

Tabela 5 – Parâmetros das Instâncias de Teste

Instância	Nomenclatura	Máquinas	Trabalhos	Operadores	Equipas de *Setup*	Data de Entrega
Pequena	P_	4	20	10	1	1 ; 2
Média	M_	4	36	10	1	1 ; 2 ; 3
Grande	G_	8	56	20	2	1 ; 2; 3
Extra Grande	E_	8	72	20	2	1 ; 2; 3

3.7. Parametrização do GRASP

A parametrização é vital para a eficácia das metaheurísticas. Parâmetros bem ajustados aumentam a velocidade de convergência e a qualidade da solução. Más escolhas podem levar a resultados abaixo dos ideais. A parametrização permite a adaptação dos algoritmos para responder a desafios específicos.

Neste trabalho em particular a parametrização foi efetuada por experimentação, selecionando-se uma combinação que reproduzia bons resultados.

Os parâmetros estudados, mencionados no subtópico 3.5.1. GRASP – *Greedy Randomized Adaptive Search Procedures*, foram os seguintes:

- α – define a dimensão da lista RCL, que será α% de CL;

- *Constructions_Max_Iterations* – número de soluções iniciais a construir;
- *LS_Max_Iterations* – número máximo de soluções correntes em que se analisa vizinhos na fase de *Local Seach* (caso não se detete um ótimo local até esse momento).

Uma vez que o GRASP foi aplicado a instâncias de diferentes dimensões, conduziu-se experimentos para encontrar boas combinações para o cenário de maior dimensão. O pressuposto assumido foi que se uma determinada combinação produzia boas soluções para as maiores dimensões, também deveria produzir para as instâncias de menores dimensões.

A metodologia de seleção da melhor combinação passou por um ajuste iterativo dos parâmetros até que os resultados obtidos fossem bons – valores de *tardiness* e soma das *makespans* reduzidos – e consistentes – após vários testes com a mesma instância, os valores não fossem significativamente distintos, mostrando uma convergência comum.

A Tabela 6 representa a combinação capaz de produzir bons resultados para a instância maior e, por isso, selecionada para todas as dimensões.

Tabela 6 – Parâmetros do GRASP

α	***Constructions_Max_Iterations***	***LS_Max_Iterations***
0,5	100	50

4. RESULTADOS E DISCUSSÃO

O capítulo 4 dedica-se à exposição dos resultados obtidos bem como à discussão das principais diferenças entre as várias abordagens e estratégias utilizadas. Assim, primeiramente será apresentada a comparação entre o *Strip-Packing* e o Time Index. Posteriormente analisam-se os resultados computacionais (gap, tempo de computação, nº de soluções ótima, a percentagem de soluções ótimas obtidas e ganhos em gap e tempo computacional) do modelo e das heurísticas matemáticas implementadas. O capítulo termina com uma avaliação ao desempenho da metaheurística (GRASP) e da heurística (RULS).

Os resultados foram obtidos através da utilização do solver *CPLEX 22.1.1.0*, com programação feita em *Python* utilizando a biblioteca *Pulp* no *IDE Microsoft Visual Studio Code 2023*. Os testes computacionais foram executados em um processador *Intel Core i7-8550U CPU 1.80GHz*, com *8.00 GB* de memória *RAM*.

Apesar de a revisão bibliográfica apontar para uma superioridade do *solver Gurobi* face ao *CPLEX*, testes experimentais realizados no início do projeto revelaram superioridade por parte do *CPLEX*. Este fenómeno poderá ser explicado pela recente atualização do *CPLEX* em 2022 – versão mais recente que os artigos apresentados.

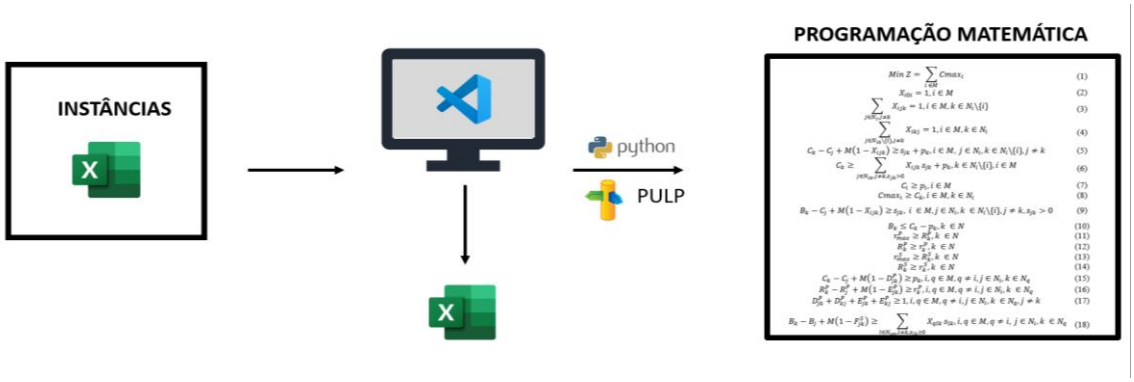


Figura 17 – Arquitetura de Dados

4.1. Comparação do *Strip-Packing* com *Time Index*

Inicialmente refletiu-se relativamente ao peso computacional de cada uma das abordagens. A Tabela 7 indica-nos o número de variáveis necessárias para a resolução de um problema de 4 máquinas e 24 tarefas (das quais 4 tarefas são iniciais).

Tabela 7 – Comparação do número de variáveis necessárias em cada abordagem

Strip-Packing	C_j	B_j	$Cmax_i$	X_{ijk}	T_j	R_j^P	R_j^S	D_{jk}^P	E_{jk}^P	F_{jk}^S	G_{jk}^S	**Total**
	24	20	4	142	24	24	20	414	414	282	282	1650
Time Index	C_j	B_j	$Cmax_i$	X_{ijk}	T_j	E_{ikt}	F_{ijkt}					**Total**
	24	24	4	142	24	69096	230320					299634

Complementarmente, o modelo *Strip-Packing* carece de 2186 restrições enquanto o modelo *Time Index* necessita de 236582 restrições. Esta análise preliminar faz antever que o modelo *Time Index*

deverá ter uma performance mais baixa face ao *Strip-Packing* pois o problema torna-se muito mais complexo e exige um maior esforço computacional.

Outro aspeto negativo do *Time Index* a destacar é o facto de o modelo não estar preparado para a possibilidade de a execução de todas as tarefas necessitar de extrapolar o período limitado pelo t_{max}.

Após correr o modelo *Time Index* por 1 hora, não se obteve qualquer solução para as instâncias desenvolvidas. Os resultados obtidos comprovaram o esperado, o modelo *Time Index* tem uma performance muito inferior ao *Strip-Packing* e, por esse motivo, análises posteriores focam-se exclusivamente no *Strip-Packing*. Os testes realizados apoiam as conclusões obtidas por (Fanjul-Peyro, 2020), que evidenciavam a supremacia do *Strip-Packing* face ao *Time Index*.

4.2. Resultados obtidos com o Modelo Matemático

Neste subcapítulo encontram-se os resultados obtidos para os testes executados com o modelo matemático desenvolvido. Além de se retirar conclusões face ao desempenho do modelo, de forma isolada, também se irão tirar conclusões quanto ao seu desempenho caso se forneça *lower bound* (que limitará o campo de soluções possíveis através do limite inferior para a soma dos *makespan* de todas as máquinas) e/ou *upper bound* (através da introdução de uma solução válida inicial).

A Tabela 8 mostra a performance da execução isolada do modelo matemático para as instâncias de pequena dimensão. Assim foi possível obter 6 soluções ótimas com um tempo médio computacional de 1538,48 segundos e um gap médio de 6,43%. No modelo de minimização da soma das *makespans*, encontram-se duas soluções sendo que o gap e tempo computacional médios são, respetivamente, de 25,71% e 2872,5 segundos.

Tabela 8 – Resultados do modelo matemático para as instâncias pequenas

	1ª Fase – Min *Tardiness*			**2ª Fase – Min Soma Cmax**			
I	Solução (*Tard.*) (min)	GAP (%)	Tempo (s)	S. Cmax Inicial (min)	Solução (S. Cmax) (min)	GAP (%)	Tempo (s)
P_1	1730	0,00	152,11	10139	9651	50,06	3600
P_2	2117	0,00	315,33	10614	10308	40,49	3600
P_3	3159	9,50	3600	11628	10982	38,07	3600
P_4	2817	41,92	3600	11948	11676	30,12	3600
P_5	7087	0,00	2534,64	11790	11183	45,42	3600
P_6	1000	0,00	268,83	10031	9126	0,00	268,41
P_7	697	0,00	115,88	11448	11068	1,48	3600
P_8	1616	0,00	1641,08	10482	10084	0,00	1111,59
Média		**6,43**	**1528,48**			**25,71**	**2872,5**

A Tabela 9 mostra os resultados obtidos com a introdução de uma solução inicial produzida pelo GRASP. Aqui é possível verificar que se encontraram 6 soluções ótimas para a componente de *tardiness* (objetivo de minimização da 1º fase), com uma média de tempo computacional inferior a

1200 segundos (20 minutos) e 5,11% de gap. Além disso, 62,5% das instâncias encontraram a solução ótima num período inferior a 254 segundos. Quanto à 2ª fase (minimização da soma das *makespans*), é possível encontrar 3 soluções ótimas embora o gap médio seja de 21,70%.

Tabela 9 – Resultados do modelo c/ *upper bound* para as instâncias pequenas

I	GRASP		1ª Fase – Min *Tardiness*			2ª Fase – Min Soma Cmax			
	Tard. (min)	S. Cmax (min)	Solução (*Tard.*) (min)	GAP (%)	Tempo (s)	S.Cmax Inicial (min)	Solução (S. Cmax)	GAP (%)	Tempo (s)
P_1	1730	9577	1730	0,00	69,42	9577	9577	39,60	3600
P_2	2117	10291	2117	0,00	65,67	10291	10271	37,76	3600
P_3	3392	11289	3159	8,64	3600	11138	10982	29,84	3600
P_4	2817	11676	2817	32,27	3600	11676	11676	25,50	3600
P_5	7485	11298	7087	0,00	217,16	11183	11183	40,90	3600
P_6	1048	9224	1000	0,00	253,81	10668	9126	0,00	349,62
P_7	719	11101	697	0,00	16,44	11101	11068	0,00	2664,42
P_8	1823	10426	1616	0,00	1581,86	10294	10084	0,00	1016,2
Média				**5,11**	**1175,55**			**21,70**	**2753,78**

Os resultados obtidos com a introdução de um *lower bound* na soma do *makespan* de todas as máquinas revelou-se positivo (Tabela 10). Assim, no 1º modelo foi possível encontrar o *tardiness* ótimo para 6 das 8 soluções (75% das vezes), com um tempo computacional médio inferior a 1600 segundos. Ainda na fase de minimização de *tardiness*, metade das soluções encontrou a solução ótima em menos de 7 minutos (420 segundos). Os resultados obtidos no 2º modelo apenas permitem encontrar 2 soluções ótimas, obtendo um gap médio de 4,16%.

Tabela 10 – Resultados do modelo c/ *lower bound* para as instâncias pequenas

I	1º Fase – Min *Tardiness*			2º Fase – Min Soma Cmax				
	Solução (*Tard.*) (min)	GAP (%)	Tempo (s)	S. Cmax Inicial (min)	*LB* (S. Cmax) (min)	Solução (S. Cmax) (min)	GAP (%)	Tempo (s)
P_1	1730	0,00	65,92	10139	8888	9355	4,99	3600
P_2	2117	0,00	291,55	10614	9722	10271	5,35	3600
P_3	3159	9,53	3600	11628	9503	10982	13,47	3600
P_4	2817	40,93	3600	11948	10997	11676	5,82	3600
P_5	7087	0,00	2616,92	11790	11014	11183	1,51	3600
P_6	1000	0,00	419,3	10031	9044	9126	0,00	948,91
P_7	697	0,00	115,41	11448	9937	11068	0,00	1160,45
P_8	1616	0,00	1995,08	10482	9870	10084	2,12	3600
Média		**6,31**	**1588,02**				**4,16**	**2963,67**

A Tabela 11 une as duas abordagens mencionadas *lower bound* (fornecido pela heurística RULS) e *upper bound* (solução inicial fornecida pelo GRASP). Na primeira fase (minimização do *tardiness*), é possível encontrar a solução ótima 75% das vezes (6 soluções ótimas). Além disso, 62,5% das instâncias encontraram a solução ótima num período inferior a 210 segundos. Na 2º fase, 3 das instâncias atingiram a solução ótima, sendo que 7 das 8 instâncias tiveram gaps inferiores a 6%.

Tabela 11 – Resultados do modelo c/ *lower bound* & *upper bound* para as instâncias pequenas

I	**GRASP**		**1º Fase – Min *Tardiness***			**2º Fase – Min Soma Cmax**				
	Tard. (min)	S. Cmax (min)	Solução (*Tard.*) (min)	GAP (%)	Tempo (s)	S. Cmax Inicial (min)	*LB* (S. Cmax) (min)	Solução (S. Cmax) (min)	GAP (%)	Tempo (s)
P_1	1730	9583	1730	0,00	160,17	9577	8888	9355	4,99	3600
P_2	2117	10271	2117	0,00	60,02	10455	9722	10271	5,35	3600
P_3	3435	11349	3159	4,05	3600	11067	9503	10982	13,47	3600
P_4	2817	12033	2817	31,13	3600	11676	10997	11676	5,82	3600
P_5	7485	11298	7087	0,00	209,07	11183	11014	11183	1,51	3600
P_6	1048	9439	1000	0,00	14,36	10123	9044	9126	0,00	629,38
P_7	719	11101	697	0,00	15,64	11101	9937	11068	0,00	1125,55
P_8	1823	10426	1616	0,00	1533,52	10378	9870	10084	0,00	1005,42
Média				**4,40**	**1149,09**				**3,89**	**2595,04**

A Tabela 12 compara os resultados em termos de gap médio, tempo computacional médio e nº de soluções ótimas obtidas, percentagem de soluções ótimas obtidas e ganhos em termos de gap e tempo computacional nas diferentes estratégias: M (apenas o modelo *Strip-Packing*), M+U (modelo *Strip-Packing* com o fornecimento de uma solução válida inicial pelo GRASP, *upper bound*), M+L (modelo *Strip-Packing* com o fornecimento de um *lower bound*, pela heurística *RULS*) e M+U+L (modelo *Strip-Packing* com *upper bound* e *lower bound*).

Primeiramente, é possível observar que o fornecimento de um *upper bound* e um *lower bound* tem um impacto muito positivo na obtenção de soluções com menor gap e menor tempo computacional. O impacto do *lower bound* é particularmente sentido na 2ª fase (modelo de minimização da soma dos *makespan* de todas as máquinas) onde é clara a força do valor obtido na heurística RULS, o que permite menores gaps (ganhos em termos de gap superiores a 83%). Por outro lado, o impacto da solução fornecida pelo GRASP (*upper bound*) sente-se particularmente na 1ª fase (modelo de minimização dos atrasos), onde as soluções de M+U e M+U+L mostram tempos médios claramente inferiores e gaps mais reduzidos. Na 1ª fase, apesar de os ganhos em M+U+L serem superiores a M+U, em ambos o *warm-start* permitiu ganhos de tempo superiores a 20% e de gap superiores a 23%.

Resumindo, todas as abordagens permitem obter o mesmo número de soluções ótimas na 1ª fase (minimização do *tardiness*), no entanto, os esforços computacionais são claramente inferiores quando é introduzida uma solução inicial (M+U e M+U+L). Compreende-se ainda que as abordagens

M+U e M+U+L são as que atingem mais soluções ótimas na 2ª fase apesar de M+L e M+U+L serem as permitem obter menores gaps.

Tabela 12 – Quadro resumo resultados modelo

		M	M+U	M+L	M+U+L
1º Fase	GAP Médio (%)	6,43	5,11	6,31	4,40
	Tempo Médio (s)	1528,48	1175,55	1588,02	1149,09
	Sol. Ótimas (Nº)	6	6	6	6
	Sol. Ótimas (%)	75	75	75	75
	GAP - Ganho médio (%)	-	20,52	1,86	31,57
	Tempo – Ganho médio (%)	-	23,09	-3,90	24,82
2º Fase	GAP Médio (%)	25,71	21,70	4,16	3,89
	Tempo Médio (s)	2872,5	2753,78	2963,67	2595,04
	Sol. Ótimas (Nº)	2	3	2	3
	Sol. Ótimas (%)	25	37,5	25	37,5
	GAP - Ganho médio (%)	-	15,60	83,82	84,87
	Tempo – Ganho médio (%)	-	4,13	-3,17	9,66

4.3. Avaliação do desempenho do GRASP

O objetivo deste capítulo é avaliar o desempenho do GRASP em duas vertentes: qualidade da solução (através da proximidade à solução ótima) e desempenho computacional (tempo necessário para obter soluções).

Para avaliar a qualidade das soluções, procedeu-se à recolha de 3 resultados do GRASP para cada uma das instâncias pequenas – Apêndice 1. Para cada instância, calculou-se os valores mínimos, máximos e médios de *tardiness* e da soma da *makespan* das máquinas, comparando-se com as soluções ótimas obtidas (Tabela 13). As principais conclusões a retirar da Tabela 13 é a boa qualidade das soluções que, em média, possui um desvio de 4,07% face ao valor ótimo de *tardiness* e 1,47% face ao valor ótimo da soma das *makespan* de todas as máquinas. Destacar que, no que diz respeito à componente de *tardiness*, a solução ótima é encontrada pelo GRASP em todos os testes executados para as instâncias 1, 2 e 4. Além disso, o GRASP foi capaz de encontrar a solução ótima (*tardiness* + soma cmax) da instância 4 no primeiro teste (Apêndice 1).

Tabela 13 – Comparação dos resultados obtidos do GRASP com a solução ótima

I	Solução GRASP						Solução Ótima		Desvio (%)	
	Tardiness (min)			Soma Cmax (min)			*Tardiness* (min)	Soma Cmax (min)	*Tardiness* (min)	Soma Cmax (min)
	Mín	Máx	Méd	Min	Máx	Méd				
P_1	1730	1730	1730,0	9577	9583	9580,0	1730	9355*	**0,00**	2,41
P_2	2117	2117	2117,0	10308	10410	10373,3	2117	10271*	**0,00**	1,00
P_3	3400	3400	3400,0	11310	11314	11311,3	3159*	10982*	7,63	3,00
P_4	2817	2817	2817,0	11676	12033	11914,0	2817*	11676*	**0,00**	2,04
P_5	7485	7485	7485,0	11298	11298	11298,0	7087	11183*	5,62	1,03
P_6	1048	1048	1048,0	9207	9418	9283,0	1000	9126	4,80	1,72
P_7	719	719	719,0	11101	11101	11101,0	697	11068	3,16	0,30
P_8	1787	1824	1799,3	10191	10217	10208,3	1616	10084*	11,34	1,23
								Média	**4,07**	**1,59**

(*) Resultados obtidos após correr o modelo da respetiva fase por um período de 6 horas.

A Tabela 14 apresenta os tempos computacionais associados aos resultados da Tabela 13. Uma rápida análise permite-nos concluir que o tempo computacional varia entre 18 e 37 segundos, sendo que a média é de 26,57 segundos. Estes são valores esperados uma vez que estamos a trabalhar um problema difícil. Além disso, o número médio de 5 tarefas por máquina (não contabilizando as tarefas iniciais) permite cerca de 120 possíveis sequências em cada máquina (120 = 5!), que será particularmente impactante na fase de *Local Search*. Como o problema é complexo, considerou-se que seria favorável optar por uma performance mais exigente do GRASP para tentar atingir a solução (quase) ótima, o que permitiria um menor esforço por parte do modelo.

Tabela 14 – Tempos Computacionais das Instâncias Pequenas

I	Tempo Computacional (s)		
	Mín	Máx	Méd
P_1	35,61	36,12	35,92
P_2	35,71	36,29	35,97
P_3	21,87	23,62	22,54
P_4	24,58	24,81	24,67
P_5	25,89	30,34	27,81
P_6	20,69	21,28	21,01
P_7	25,37	26,56	25,94
P_8	18,53	18,97	18,73
		Média	**26,57**

Tal como previamente explorado no capítulo 3.7. Parametrização do GRASP, como não existe soluções ótimas para as instâncias médias, grandes e extra grandes com as quais comparar os

resultados obtidos, é importante executar vários testes para identificar uma combinação de parâmetros que, corrida várias vezes, apresente resultados relativamente estáveis. Neste sentido, a Tabela 15 apresenta os resultados obtidos para 3 testes em cada uma das 8 instâncias médias. Como é possível observar na Tabela 15, os tempos computacionais variam entre 444 e 590 segundos, valores muito superiores aos obtidos na Tabela 13. Era expectável que o valor subisse porque o problema tornou-se maior, atualizando o número de tarefas para 40 (das quais 4 são tarefas iniciais) e 3 dias como possíveis para entrega. Assim, cada máquina terá, em média, 9 tarefas afetas o que contabiliza 362880 possíveis sequências (362880 = 9!) em apenas 1 máquina.

Tabela 15 – Resultados e tempos computacionais das instâncias médias

I	**Solução 1**			**Solução 2**			**Solução 3**		
	Tempo (s)	*Tard.* (min)	S. Cmax (min)	Tempo (s)	*Tard.* (min)	S. Cmax (min)	Tempo (s)	*Tard.* (min)	S. Cmax (min)
M_1	484,42	3479	15274	479,12	3269	15871	476,62	3401	15499
M_2	455,06	12761	17622	442,08	12324	17378	456,03	12412	17142
M_3	458,19	20295	24090	466,02	20166	23789	460,13	19162	23694
M_4	506,95	13151	20212	514,05	13774	20484	500,98	13903	20499
M_5	458,11	16130	21671	468,19	16084	21403	444,16	16155	21372
M_6	567,10	12141	20492	550,89	12101	20507	550,77	12072	20532
M_7	589,23	17636	21263	585,56	17742	21236	546,72	18805	21343
M_8	478,45	25096	21336	479,47	23568	21609	483,75	24056	21430

A Tabela 16 sumaria os resultados da Tabela 15. Assim, concluiu-se que o tempo computacional médio é de 495,92 segundos, sendo que 75% dos tempos médios são inferiores a 540 segundos (9 minutos). Além disso, os resultados evidenciam estabilidade, isto é, a diferença entre os valores máximos e mínimos não é muito significativa, o que poderá significar que encontramos boas soluções iniciais para o problema.

Tabela 16 – Métricas para as instâncias médias

I	**Tempo Computacional (s)**			***Tardiness* (min)**			**Soma Cmax (min)**		
	Mín	Máx	Méd	Mín	Máx	Méd	Mín	Máx	Méd
M_1	476,62	484,42	480,05	3269	3479	3383,00	15274	15871	15548,00
M_2	442,08	456,03	451,06	12324	12761	12499,00	17142	17622	17380,67
M_3	458,19	466,02	461,45	19162	20295	19874,33	23694	24090	23857,67
M_4	500,98	514,05	507,33	13151	13903	13609,33	20212	20499	20398,33
M_5	444,16	468,19	456,82	16084	16155	16123,00	21372	21671	21482,00
M_6	550,77	567,10	556,25	12072	12141	12104,67	20492	20532	20510,33
M_7	546,72	589,23	573,84	17636	18805	18061,00	21236	21343	21280,67
M_8	478,45	483,75	480,56	23568	25096	24240,00	21336	21609	21458,33
		Média	**495,92**						

A Tabela 17 mostra que os tempos computacionais para as instâncias grandes variam entre 968 e 1996 segundos (ou seja, entre 16 e 33 minutos), valores cerca 3 vezes superiores aos obtidos na Tabela 15. Nesta instância, cada máquina tem, em média, 7 tarefas afetas, o que contabiliza 30240 possíveis sequências (30240 = 7!) em cada uma delas. Este é um valor inferior ao número de possíveis sequências em cada máquina na instância média. No entanto, estas instâncias serão mais complexas que as médias porque tem mais tarefas, máquinas e recursos, que adensam significativamente a dificuldade de resolução do problema.

Tabela 17 – Resultados e tempos computacionais das instâncias grandes

I	**Solução 1**			**Solução 2**			**Solução 3**		
	Tempo (s)	*Tard.* (min)	S. Cmax (min)	Tempo (s)	*Tard.* (min)	S. Cmax (min)	Tempo (s)	*Tard.* (min)	S. Cmax (min)
G_1	968,37	6111	25543	1011,71	6211	25681	1035,66	6071	25572
G_2	1629,15	3321	24532	1664,27	3321	24567	1731,36	3321	24539
G_3	1198,52	2047	24603	1126,42	2076	24566	1178,86	2047	25586
G_4	1463,13	2689	23666	1496,12	2689	23666	1484,84	2689	23666
G_5	1305,28	2138	24863	1224,11	2138	24725	1286,04	2138	24927
G_6	1512,47	4985	28574	1421,65	4704	29380	1494,48	4796	29956
G_7	1964,56	2835	22708	1953,17	2835	22734	1995,30	2828	24602
G_8	1683,00	4566	25128	1686,36	4566	25062	1628,24	4566	26842

A Tabela 18 resume os resultados da Tabela 17, concluindo-se que o tempo computacional médio é de 1464,29 segundos (cerca de 24 minutos), sendo que 62,5% dos resultados são obtidos em menos de 1500 segundos (25 minutos). Mais uma vez, os resultados evidenciam estabilidade, sendo a diferença entre os valores máximos e mínimos não muito significativa.

Tabela 18 – Métricas para as instâncias grandes

I	**Tempo Computacional (s)**			***Tardiness* (min)**			**Soma Cmax (min)**		
	Mín	Máx	Méd	Mín	Máx	Méd	Mín	Máx	Méd
G_1	968,37	1035,66	1005,25	6071	6211	6131,00	25543	25681	25598,67
G_2	1629,15	1731,36	1674,93	3321	3321	3321,00	24532	24567	24546,00
G_3	1126,42	1198,52	1167,93	2047	2076	2056,67	24566	25586	24918,33
G_4	1463,13	1496,12	1481,36	2689	2689	2689,00	23666	23666	23666,00
G_5	1224,11	1305,28	1271,81	2138	2138	2138,00	24725	24927	24838,33
G_6	1421,65	1512,471	1476,20	4704	4985	4828,33	28574	29956	29303,33
G_7	1953,17	1995,3	1971,01	2828	2835	2832,67	22708	24602	23348,00
G_8	1628,24	1686,36	1665,87	4566	4566	4566,00	25062	26842	25677,33
		Média	**1464,29**						

A Tabela 19 mostra que os tempos computacionais para as instâncias grandes variam entre 1813 e 3037 segundos (ou seja, entre cerca de 30 e 50 minutos), com valores aproximadamente 1,7 vezes superiores aos obtidos na Tabela 17.

Nesta instância, cada máquina tem, em média, 9 tarefas afetas, à semelhança da instância média. No entanto, em termos de número de máquinas e recursos, esta equipara-se à instância grande. Além disso, possui mais 16 tarefas que a instância grande (80 tarefas, das quais 8 tarefas iniciais), o que permite afirmar que é a maior instância em estudo. Neste sentido, o aumento de tempo computacional era expectável.

Tabela 19 – Resultados e tempos computacionais das instâncias extra grandes

I	**Solução 1**			**Solução 2**			**Solução 3**		
	Tempo (s)	*Tard.* (min)	S. Cmax (min)	Tempo (s)	*Tard.* (min)	S. Cmax (min)	Tempo (s)	*Tard.* (min)	S. Cmax (min)
E_1	2986,64	16447	35180	2986,03	16991	35732	2714,72	16322	35056
E_2	2562,06	15750	33688	2538,82	15801	31986	2491,97	15437	34353
E_3	2061,07	14791	31647	2329,21	14873	31688	2365,18	14944	31759
E_4	1979,41	26133	34374	1982,18	23852	33402	1813,48	25952	33492
E_5	2389,69	19884	33577	2499,98	19600	33921	2487,83	20656	33747
E_6	2794,36	9926	32073	3036,04	10755	32005	2825,99	10228	32375
E_7	2802,37	5385	33489	2955,83	4126	33059	2593,42	5369	33617
E_8	2114,03	12282	33415	2356,82	11628	32585	2276,64	12694	32947

A Tabela 20 permite analisar outros aspetos como o tempo computacional médio de 2497,66 segundos (cerca de 42 minutos), sendo que 62,5% dos resultados são obtidos em menos de 2700 segundos (45 minutos). Mais uma vez, os resultados evidenciam estabilidade, não existindo grande discrepância entre os valores máximos e mínimos.

Tabela 20 – Métricas para as instâncias extra grandes

I	**Tempo Computacional (s)**			***Tardiness* (min)**			**Soma Cmax (min)**		
	Mín	Máx	Méd	Mín	Máx	Méd	Mín	Máx	Méd
E_1	2714,72	2986,64	2895,80	16322	16991	16586,67	35056	35732	35322,67
E_2	2491,97	2562,06	2530,95	15437	15801	15662,67	31986	34353	33342,33
E_3	2061,07	2365,18	2251,82	14791	14944	14869,33	31647	31759	31698,00
E_4	1813,48	1982,18	1925,02	23852	26133	25312,33	33402	34374	33756,00
E_5	2389,69	2499,98	2459,17	19600	20656	20046,67	33577	33921	33748,33
E_6	2794,36	3036,04	2885,46	9926	10755	10303,00	32005	32375	32151,00
E_7	2593,42	2955,83	2783,87	4126	5385	4960,00	33059	33617	33388,33
E_8	2114,03	2356,82	2249,16	11628	12694	12201,33	32585	33415	32982,33
		Média	**2497,66**						

4.4. Avaliação do desempenho da heurística RULS

O presente capítulo tem como principal objetivo medir a força do *lower bound* fornecido ao modelo. Tal como previamente mencionado, esta heurística irá devolver um *lower bound* para a soma dos *makespan* de todas as máquinas. Para se puder estabelecer uma comparação, foi adicionada uma coluna com os valores obtidos pela relaxação linear do modelo da 2ª fase. Salientar que todas as outras condições se mantiveram, inclusive o fornecimento de uma solução inicial da 1º fase (modelo que minimiza o *tardiness*).

Pela análise da Tabela 21, facilmente se compreende que os valores obtidos são muito positivos uma vez que se aproximam daquela que será a solução ótima do modelo, tendo um desvio médio de 5,55% e um desvio máximo inferior a 13,5%. Além disso, a superioridade da heurística face à relaxação linear é evidente, destacando-se ao obter valores de LB cerca de 3 vezes superiores.

Tabela 21 – Comparação dos resultados obtidos da Heurística RULS com a solução ótima

I	Relaxação Linear (min)	*LB* (min)	Solução Ótima → Soma Cmax (min)	Desvio (%)
P_1	3053,06	8888	9355*	4,99
P_2	3385,57	9722	10271*	5,35
P_3	3244,06	9503	10982*	13,47
P_4	3770,04	10997	11676*	5,82
P_5	3735,03	11014	11183*	1,51
P_6	3194,49	9044	9126	0,90
P_7	3403,34	9937	11068	10,22
P_8	3451,74	9870	10084*	2,12
			Média	**5,55**

(*) Resultados obtidos após correr o modelo da respetiva fase por um período de 6 horas.

A Tabela 22, 23 e 24 refletem os resultados obtidos com a heurística RULS que resultariam como *lower bound* para as instâncias médias, grandes e extra grandes. Como não se conhece a solução ótima para estas instâncias, mas pretendia-se avaliar a qualidade do *lower bound*, optou-se por comparar a sua solução da heurística RULS com a melhor solução obtida pelo GRASP. Note-se que a solução ótima poderá obrigar a um valor mais elevado de Soma Cmax (como consequência de uma diminuição de *tardiness*), ao que seriam necessários testes adicionais para conferir a qualidade do *lower bound*.

No entanto, dos resultados possíveis, a Tabela 22 mostra que a heurística do *lower bound* parece continuar muito adequada para responder ao propósito do problema, visto que a diferença entre o *lower bound* e a componente da soma das *makespans* tem, em média, uma diferença de 5,89%.

Tabela 22 – *Lower bound* para instâncias médias e comparação com solução do GRASP

I	Melhor Solução GRASP		*LB* (min)	Desvio (%)
	Tardiness (min)	Soma Cmax (min)		
M_1	3269	15871	14617	7,90
M_2	12324	17378	15630	10,06
M_3	19162	23694	22337	5,73
M_4	13151	20212	18710	7,43
M_5	16084	21403	20681	3,37
M_6	12072	20532	19971	2,73
M_7	17636	21263	20083	5,55
M_8	23568	21609	20672	4,34
			Média	**5,89**

Relativamente aos resultados da Tabela 23, estes mostram-se promissores visto que revelam proximidade entre o *lower bound* e os valores obtidos pela heurística (componente Soma Cmax), tendo um gap médio de 4,54%.

Tabela 23 – *Lower bound* para instâncias grandes e comparação com solução do GRASP

I	Melhor Solução GRASP		*LB* (min)	Desvio (%)
	Tardiness (min)	Soma Cmax (min)		
G_1	6071	25572	24795	3,04%
G_2	3321	24532	24088	1,81%
G_3	2047	24603	23738	3,52%
G_4	2689	23666	23259	1,72%
G_5	2138	24725	24346	1,53%
G_6	4704	29380	26193	10,85%
G_7	2828	24602	21857	11,16%
G_8	4566	25062	24382	2,71%
			Média	**4,54%**

Os resultados obtidos para as instâncias extra grandes não foram diferentes. Como podemos ver na Tabela 24, a distância entre o *lower bound* e a melhor solução de GRASP é, em média, de 6,81%. Mais uma vez, este poderá ser um sinal positivo face à qualidade de soluções de ambas as estratégias.

Tabela 24 – *Lower bound* para instâncias extra grandes e comparação com solução GRASP

I	Melhor Solução GRASP		*LB* (min)	Desvio (%)
	Tardiness (min)	Soma Cmax (min)		
G_1	15013	33937	30156	11,14
G_2	15437	34353	30514	11,18
G_3	14791	31647	30051	5,04
G_4	25952	33492	32574	2,74
G_5	19884	33577	31758	5,42
G_6	9926	32073	30510	4,87
G_7	4126	33059	30180	8,71
G_8	11628	32585	30825	5,40
			Média	**6,81**

No que toca ao tempo de execução desta heurística, para todas as dimensões de instâncias, é sempre inferior a 1 segundo.

5. CONCLUSÃO

Mais do que nunca, uma empresa para ter (e manter) o sucesso deve procurar identificar e extinguir as suas ineficiências. Muitas vezes, esta tarefa carece do tratamento de informação complexa e requer sistemas de apoio à decisão que auxiliem na tomada de boas decisões, permitindo que sejam feitam de forma fundamentada.

Neste sentido, este projeto surge do desafio da INPLAS ao centro de investigação CPLOT, que procura um sistema de apoio à decisão da programação de máquinas paralelas dedicadas com *setups* dependentes da sequência de famílias, recursos adicionais e datas de entrega. Além disso, temos em mãos uma indústria que funciona com fluxo contínuo, tornando relevante o conhecimento das configurações iniciais nas máquinas em cada dia.

A resolução deste problema levanta duas principais questões para investigação. A primeira questão prende-se com a descoberta do modelo matemático que melhor resolve o problema, *Strip-Packing* ou *Time Index*. A segunda questão foca-se no estudo do impacto das heurísticas matemáticas na eficiência do algoritmo.

O presente trabalho fornece então várias contribuições ao nível do modelo matemático e da metaheurística explorada. No que reflete ao modelo matemático, identificam-se 5 contribuições. O primeiro aspeto a destacar é a adaptação do modelo geral de referência para o caso particular das máquinas paralelas dedicadas, que como já possui alocação realizada à priori, permite a simplificação de variáveis e parâmetros. Além disso, o facto de se incorporar datas de entrega torna o problema mais complexo carecendo da introdução de uma nova variável (o atraso – ou *tardiness* – de cada tarefa) e adaptação da função objetivo (que agora terá como objetivo principal a minimização do *tardiness* total). Indústrias que funcionam de forma contínua podem possuir tarefas que apesar de iniciar num determinado dia, se prolongam no seguinte. Esta tarefas são intituladas de tarefas iniciais e procura-se garantir que terão prioridade de execução perante todas as tarefas que estão previstas de ser processadas na mesma máquina. Também se compreendeu que o modelo de referência não estava preparado para a existência de tarefas que, por serem da mesma família da tarefa previamente processada, não careciam de *setup*. Nesse sentido, procedeu-se à generalização do modelo para lidar com *setups* dependentes da sequência de famílias. Por fim, desenvolveu-se uma função multi-objetivo. Estando a trabalhar com datas de entrega, o objetivo principal passa pela minimização dos atrasos. No entanto, este objetivo isolado permitia obter um vasto leque de soluções, sendo que parte destas poderiam desencadear tempos ociosos e/ou operações de *setup* não necessárias. Assim, objetivando-se uma solução que minimize o *tardiness* mas também seja compacta, introduziu-se um segundo objetivo: minimização da soma dos *makespan* de todas as máquinas.

No que diz respeito ao GRASP, a metaheurística desenvolvida, o algoritmo de referência não estava preparado para a existência de recursos de processamento, o que levou à necessidade de adaptação de toda a metaheurística, com particular cuidado no sistema de reparação. Por outro lado, como o objetivo deste estudo era, em primeiro lugar, a minimização do *tardiness,* a forma como estava a ser calculado o custo de sequenciação e timing das tarefas não era o mais adequado, obrigando a alterá-lo.

Neste trabalho são também desenvolvidas heurísticas matemáticas para atingir melhores resultados visto que o modelo matemático apenas se mostrou capaz de resolver instâncias de pequena dimensão. Uma das estratégias aplicadas foi o *warm-start* que utilizava as soluções provenientes da metaheurítica GRASP como solução inicial para o modelo desenvolvido, atuando como um *upper bound.* Por outro lado, a também desenvolvida heurística RULS fornece uma solução que atua como *lower bound* para o modelo matemático, limitando assim os possíveis valores da soma dos *makespan* da soma de todas as máquinas.

À semelhança do esperado, o fornecimento do *upper bound* e do *lower bound* teve um impacto muito positivo nos resultados obtidos. Apesar de individualmente também se terem mostrado vantajosos, a junção de ambos permitiu obter os melhores tempos computacionais e valores de gap, concluindo assim que a junção da heurística RULS e da metaheurítica GRASP ao modelo matemático é muito promissora. Salientar que as heurísticas matemáticas foram apenas aplicadas às instâncias pequenas. A 1ª fase, que pretendia minimizar o atraso (*tardiness*), alcançou a solução ótima em 75% dos testes, com uma gap médio de 4,40% em relação à solução ótima. Além disso, permitiu uma redução do gap médio em 31,57% e uma melhoria de 24,82% no tempo computacional face ao modelo isolado. A 2ª fase, que procura minimizar a soma do makespan de todas as máquinas, também apresentou resultados positivos, atingindo a solução ótima em 37,5% dos testes, obtendo um gap médio de 3,89%. Ainda na 2ª fase, os ganhos, comparativamente com o modelo original, foram de 84,87% em termos de gap e de 9,66% em tempos computacionais.

Tal como já se poderia aferir pelos bons resultados obtidos pelo modelo quando combinado com o *lower bound* fornecido pela heurística RULS e o *upper bound* fornecido pela meta-heurística GRASP, o estudo individual de cada um deles perimite confirmar a qualidade das soluções fornecidas por cada um deles. Quando comparado com a solução ótima, o resultado do GRASP é muito positivo sendo que o desvio médio do *tardiness* face à solução ótima, no pior caso foi cerca de 12%, em média 4,4%, tendo atingindo a solução ótima em 37,5% das vezes. Além da componente de *tardiness*, o GRASP também revelou qualidade na aproximação à soma dos tempos de conclusão das máquinas, com um desvio médio de 1,59%. No que diz respeito à heurística RULS, esta mostrou-se muito mais forte que as soluções obtidas por relaxação linear, o que demonstra a importância de apostar nesta estratégia e não ficar apenas dependente dos cortes internos do CPLEX. Ainda relativamente à heurística RULS, em média, o seu desvio face à solução ótima (componente soma dos *makespans* de todas as máquinas) foi de 5,55%.

O estudo aos limites computacionais do GRASP demonstrou que este é capaz de responder a problemas pequenos em tempos computacionais muito rápidos (cerca de 27 segundos). Quando estamos perante instâncias médias (neste caso até 4 máquinas e 40 tarefas) os tempos computacionais foram razoáveis (cerca de 8 minutos). No entanto, o GRASP torna-se obsoleto para instâncias de maior dimensão, necessitando de, em média, cerca de 25 e 42 minutos para obter boas soluções em instâncias grandes e extra grandes, respetivamente. Apesar de obter resultados animadores em termos de solução, o tempo computacional não razoável faz com que esta metaheurística não seja a mais adequada para problemas de elevada dimensão. Por outro lado, a heurística RULS, mesmo para instâncias de maior dimensão, permanece muito rápida – tempos computacionais inferiores a 1 segundo para todas as dimensões de instâncias.

É importante mencionar que parte deste trabalho foi apresentada na 32ª edição da conferência internacional *Flexible Automation and Intelligent Manufacturing* e aceita para publicação em uma revista indexada. O artigo resultante desse trabalho pode ser encontrado no Anexo A.

5.1. Limitações e investigação futura

Trabalhos futuros dever-se-ão focar na melhoria do desempenho do GRASP, visto que existe margem para melhorias que não foram possíveis de executar no período do projeto, permitindo assim o fornecimento de soluções iniciais em tempos computacionais mais razoáveis. Além disso, dever-se-á apostar noutro tipo de estratégias de heurísticas matemáticas, nomeadamente estratégias que permitam o fornecimento de soluções do modelo para uma metaheurísitca, sendo particularmente favorável em instâncias que permanecem com o valor de gap rígido durante parte significativa do período computacional. Por fim, poderá ser interessante desenvolver uma heurística/metaheurística que permita definir um *lower bound* para a componente *tardiness*.

Trabalhar na vertente de investigação é muito exigente/inconstante e a principal dificuldade deste trabalho relaciona-se com este aspeto. A imprevisibilidade, o processo de tentativa-erro e a necessidade de mudar o rumo de pesquisa após tempo e dedicação investidos no estudo de uma estratégia/abordagem é o principal desafio em mãos. No que diz respeito ao trabalho na sua dimensão, o aspeto mais desafiante foi o desenvolvimento de uma metaheurítica para um problema tão complexo como é a programação de máquinas paralelas dedicadas com famílias de *setup,* recursos adicionais e datas de entrega.

REFERÊNCIAS BIBLIOGRÁFICAS

Aiex, R. M., Binato, S., & Resende, M. G. C. (2003). Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, *29*(4), 393–430. https://doi.org/https://doi.org/10.1016/S0167-8191(03)00014-0

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, *246*(2), 345–378. https://doi.org/10.1016/J.EJOR.2015.04.004

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, *27*(2), 219–239. https://doi.org/10.1016/S0305-0483(98)00042-5

Allahverdi, A., & Soroush, H. M. (2008). The significance of reducing setup times/setup costs. *European Journal of Operational Research*, *187*(3), 978–984. https://doi.org/10.1016/J.EJOR.2006.09.010

Avalos-Rosales, O., Angel-Bello, F., & Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *International Journal of Advanced Manufacturing Technology*, *76*(9–12), 1705–1718. https://doi.org/10.1007/S00170-014-6390-6/METRICS

Bektur, G., & Saraç, T. (2019). A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server. *Computers and Operations Research*, *103*, 46–63. https://doi.org/10.1016/j.cor.2018.10.010

Binato, S., Hery, W. J., Loewenstern, D. M., & Resende, M. G. C. (2002). A Grasp for Job Shop Scheduling. *Essays and Surveys in Metaheuristics*, *15*, 59–79. https://link.springer.com/chapter/10.1007/978-1-4615-1507-4_3

Bitar, A., Dauzère-Pérès, S., & Yugma, C. (2021). Unrelated parallel machine scheduling with new criteria: Complexity and models. *Computers & Operations Research*, *132*, 105291. https://doi.org/10.1016/J.COR.2021.105291

Błażewicz, J., Brauner, N., & Finke, G. (2004). Scheduling with discrete resource constraints. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*.

Błażewicz, J., Kubiak, W., Röck, H., & Szwarcfiter, J. (1987). Minimizing mean flow-time with parallel processors and resource constraints. *Acta Informatica*, *24*(5), 513–524. https://doi.org/10.1007/BF00263292

Cheng, X., Feng, S., Zheng, H., Wang, J., & Liu, S. (2022). A hierarchical model in short-term hydro scheduling with unit commitment and head-dependency. *Energy*, *251*, 123908. https://doi.org/10.1016/J.ENERGY.2022.123908

Coutinho, C. P. (2011). Metodologias de Investigação em Ciências Humanas. In *Almedina*.

Dang, Q. V., van Diessen, T., Martagan, T., & Adan, I. (2021). A matheuristic for parallel machine scheduling with tool replacements. *European Journal of Operational Research*, *291*(2), 640–660. https://doi.org/10.1016/J.EJOR.2020.09.050

Daniels, R. L., Hoopes, B. J., & Mazzola, J. B. (1996). Scheduling Parallel Manufacturing Cells with Resource Flexibility. *Https://Doi.Org/10.1287/Mnsc.42.9.1260*, *42*(9), 1260–1276. https://doi.org/10.1287/MNSC.42.9.1260

Daniels, R. L., Hua, S. Y., & Webster, S. (1999). Heuristics for parallel-machine flexible-resource scheduling problems with unspecified job assignment. *Computers & Operations Research*, *26*(2), 143–155. https://doi.org/10.1016/S0305-0548(98)00054-9

de Abreu, A. P., & Fuchigami, H. Y. (2022). An efficiency and robustness analysis of warm-start mathematical models for idle and waiting times optimization in the flow shop. *Computers & Industrial Engineering*, *166*, 107976. https://doi.org/10.1016/J.CIE.2022.107976

Della Croce, F., Salassa, F., & T'Kindt, V. (2014). A hybrid heuristic approach for single machine scheduling with release times. *Computers and Operations Research*, *45*, 7–11. https://doi.org/10.1016/j.cor.2013.11.016

Edis, E. B., & Oguz, C. (2012). Parallel machine scheduling with flexible resources. *Computers & Industrial Engineering*, *63*(2), 433–447. https://doi.org/10.1016/J.CIE.2012.03.018

Edis, E. B., Oguz, C., & Ozkarahan, I. (2013). Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research*, *230*(3), 449–463. https://doi.org/10.1016/J.EJOR.2013.02.042

Edis, E. B., & Ozkarahan, I. (2012). Solution approaches for a real-life resource-constrained parallel machine scheduling problem. *International Journal of Advanced Manufacturing Technology*, *58*(9–12), 1141–1153. https://doi.org/10.1007/S00170-011-3454-8/METRICS

Fanjul-Peyro, L. (2020). Models and an exact method for the Unrelated Parallel Machine scheduling problem with setups and resources. *Expert Systems with Applications: X*, *5*, 100022. https://doi.org/10.1016/J.ESWAX.2020.100022

Fanjul-Peyro, L., Perea, F., & Ruiz, R. (2017). Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, *260*(2), 482–493. https://doi.org/10.1016/J.EJOR.2017.01.002

Fanjul-Peyro, L., Ruiz, R., & Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers & Operations Research*, *101*, 173–182. https://doi.org/10.1016/J.COR.2018.07.007

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*(2), 67–71. https://doi.org/https://doi.org/10.1016/0167-6377(89)90002-3

Feo, T. A., Sarathy, K., & McGahan, J. (1996). A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Computers & Operations*

Research, *23*(9), 881–895. https://doi.org/https://doi.org/10.1016/0305-0548(95)00084-4

Feo, T. A., Venkatraman, K., & Bard, J. F. (1991). A GRASP^{TM} for a difficult single machine scheduling problem. *Computer & Operations Research*, *18*(8), 635–643. https://doi.org/https://doi.org/10.1016/0305-0548(91)90001-8

Fernandes, J., Machado, C. F., & Amaral, L. (2020). *Methodology Used for Determination of Critical Success Factors in Adopting the New General Data Protection Regulation in Higher Education Institutions*. 71–109. https://doi.org/10.1007/978-3-030-40896-1_4

Fleszar, K., & Hindi, K. S. (2018). Algorithms for the unrelated parallel machine scheduling problem with a resource constraint. *European Journal of Operational Research*, *271*(3), 839–848. https://doi.org/10.1016/j.ejor.2018.05.056

Grigoriev, A., Sviridenko, M., & Uetz, M. (2005). Unrelated Parallel Machine Scheduling with Resource Dependent Processing Times. *Integer Programming and Combinatorial Optimization: 11th International IPCO Conference*, 182–195. https://link.springer.com/chapter/10.1007/11496915_14

Grigoriev, A., Sviridenko, M., & Uetz, M. (2007). Machine scheduling with resource dependent processing times. *Mathematical Programming*, *110*(1), 209–228. https://doi.org/10.1007/s10107-006-0059-3

Guinet, A. (1993). Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. *Http://Dx.Doi.Org/10.1080/00207549308956810*, *31*(7), 1579–1594. https://doi.org/10.1080/00207549308956810

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2013). The Distributed Assembly Permutation Flowshop Scheduling Problem. *Http://Dx.Doi.Org/10.1080/00207543.2013.807955*, *51*(17), 5292–5308. https://doi.org/10.1080/00207543.2013.807955

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2016). Heuristics for a Distributed Parallel Machine Assembly Scheduling Problem with eligibility constraints. *Proceedings of 2015 International Conference on Industrial Engineering and Systems Management, IEEE IESM 2015*, 145–153. https://doi.org/10.1109/IESM.2015.7380149

Heinz, V., Novák, A., Vlk, M., & Hanzálek, Z. (2022). Constraint Programming and constructive heuristics for parallel machine scheduling with sequence-dependent setups and common servers. *Computers & Industrial Engineering*, *172*, 108586. https://doi.org/10.1016/J.CIE.2022.108586

Henrique, T., Pinheiro, G., Santos, A., & Venâncio, C. R. (2014). A hybrid Lagrangean metaheuristic for single machine scheduling problem with sequence-dependent setup times and due dates. *Comnputer & Operations Research*.

Inplas - Simoldes. (n.d.). Retrieved December 19, 2022, from https://www.simoldes.com/plastics-companies/inplas/

Kellerer, H., & Strusevich, V. A. (2003). Scheduling parallel dedicated machines under a single non-shared resource. *European Journal of Operational Research*, *147*(2), 345–364. https://doi.org/10.1016/S0377-2217(02)00246-1

M. Pour, S., Drake, J. H., Ejlertsen, L. S., Rasmussen, K. M., & Burke, E. K. (2018). A hybrid Constraint Programming/Mixed Integer Programming framework for the preventive signaling maintenance crew scheduling problem. *European Journal of Operational Research*, *269*(1), 341–352. https://doi.org/10.1016/J.EJOR.2017.08.033

Mittelmann, H. (2022). *Decison Tree for Optimization Software - INFORM 2022*. http://plato.asu.edu/bench.html

Nguyen, V. H., Tuong, N. H., Tran, V. H., & Thoai, N. (2013). An MILP-based makespan minimization model for single-machine scheduling problem with splitable jobs and availability constraints. *2013 International Conference on Computing, Management and Telecommunications, ComManTel 2013*, 397–400. https://doi.org/10.1109/COMMANTEL.2013.6482427

Ólafsson, S., & Shi, L. (2000). A method for scheduling in parallel manufacturing systems with flexible resources. *IIE Transactions 2000 32:2*, *32*(2), 135–146. https://doi.org/10.1023/A:1007610330810

Ozer, E. A., & Sarac, T. (2019). MIP models and a matheuristic algorithm for an identical parallel machine scheduling problem under multiple copies of shared resources constraints. *TOP*, *27*(1), 94–124. https://doi.org/10.1007/S11750-018-00494-X/TABLES/11

Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers & Operations Research*, *35*(10), 3202–3212. https://doi.org/10.1016/J.COR.2007.02.014

Pinedo, M. L. (1995). Scheduling: Theory, algorithms, and systems, fifth edition. *Scheduling: Theory, Algorithms, and Systems, Fifth Edition*, 1–670. https://doi.org/10.1007/978-3-319-26580-3/COVER

Pinto, J. M., & Grossmann, I. E. (1997). A logic-based approach to scheduling problems with resource constraints. *Computers and Chemical Engineering*, *21*(8), 801–818. https://doi.org/10.1016/S0098-1354(96)00318-3

Puchinger, J., & Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. *Lecture Notes in Computer Science*, *3562*(PART II), 41–53. https://doi.org/10.1007/11499305_5/COVER

Rajkumar, M., Asokan, P., Anilkumar, N., & Page, T. (2011). A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints. *International Journal of Production Research*, *49*(8), 2409–2423. https://doi.org/https://doi.org/10.1080/00207541003709544

Reklaitis, G. V. (1996). Overview of Scheduling and Planning of Batch Process Operations. *Batch Processing Systems Engineering*, 660–705. https://doi.org/10.1007/978-3-642-60972-5_27

Simoldes Plastics. (n.d.). Retrieved December 19, 2022, from https://www.simoldes.com/plastics/inicio/

Sioud, A., Gravel, M., & Gagné, C. (2012). A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times. *Computers and Operations Research*, *39*(10), 2415–2424. https://doi.org/10.1016/j.cor.2011.12.017

Slowinski, R. (1980). Two Approaches to Problems of Resource Allocation Among Project Activities — A Comparative Study. *Https://Doi.Org/10.1057/Jors.1980.134*, *31*(8), 711–723. https://doi.org/10.1057/JORS.1980.134

Soares, L. C. R., & Carvalho, M. A. M. (2022). Application of a hybrid evolutionary algorithm to resource-constrained parallel machine scheduling with setup times. *Computers & Operations Research*, *139*. https://doi.org/https://doi.org/10.1016/j.cor.2021.105637

Stockemer, D. (2019). Quantitative Methods for the Social Sciences. In *Quantitative Methods for the Social Sciences*. Springer International Publishing. https://doi.org/10.1007/978-3-319-99118-4

Tavakkoli-Moghaddam, R., Taheri, F., Bazzazi, M., Izadi, M., & Sassani, F. (2009). Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. *Computers & Operations Research*, *36*(12), 3224–3230. https://doi.org/10.1016/J.COR.2009.02.012

Tran, T. T., Araujo, A., & Beck, J. C. (2016). Decomposition Methods for the Parallel Machine Scheduling Problem with Setups. *Https://Doi.Org/10.1287/Ijoc.2015.0666*, *28*(1), 83–95. https://doi.org/10.1287/IJOC.2015.0666

Vahedi-Nouri, B., Tavakkoli-Moghaddam, R., Hanzálek, Z., & Dolgui, A. (2022). Workforce planning and production scheduling in a reconfigurable manufacturing system facing the COVID-19 pandemic. *Journal of Manufacturing Systems*, *63*, 563–574. https://doi.org/10.1016/J.JMSY.2022.04.018

Vakhania, N., Hernandez, J. A., & Werner, F. (2014). Scheduling unrelated machines with two types of jobs. *Https://Doi.Org/10.1080/00207543.2014.888789*, *52*(13), 3793–3801. https://doi.org/10.1080/00207543.2014.888789

Vallada, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, *211*(3), 612–622. https://doi.org/10.1016/J.EJOR.2011.01.011

Ventura, J. A., & Kim, D. (2000). Parallel machine scheduling about an unrestricted due date and additional resource constraints. *IIE Transactions (Institute of Industrial Engineers)*, *32*(2), 147–153. https://doi.org/10.1023/A:1007662314880/METRICS

Villa, F., Vallada, E., & Fanjul-Peyro, L. (2018). Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource. *Expert Systems with Applications*, *93*, 28–38. https://doi.org/10.1016/J.ESWA.2017.09.054

Wang, S., Wu, R., Chu, F., & Yu, J. (2020). Identical parallel machine scheduling with assurance of maximum waiting time for an emergency job. *Computers & Operations Research*, *118*, 104918. https://doi.org/10.1016/J.COR.2020.104918

Yepes-Borrero, J. C., Perea, F., Ruiz, R., & Villa, F. (2021). Bi-objective parallel machine scheduling with additional resources during setups. *European Journal of Operational Research*, *292*(2), 443–455. https://doi.org/10.1016/J.EJOR.2020.10.052

Yepes-Borrero, J. C., Villa, F., Perea, F., & Caballero-Villalobos, J. P. (2020). GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Applications*, *141*, 112959. https://doi.org/10.1016/J.ESWA.2019.112959

Yunusoglu, P., & Yildiz, S. T. (2022). Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, *60*(7), 2212–2229. https://doi.org/10.1080/00207543.2021.1885068/SUPPL_FILE/TPRS_A_1885068_SM9970.RAR

Zhang, L., Deng, Q., Lin, R., Gong, G., & Han, W. (2021). A combinatorial evolutionary algorithm for unrelated parallel machine scheduling problem with sequence and machine-dependent setup times, limited worker resources and learning effect. *Expert Systems with Applications*, *175*, 114843. https://doi.org/10.1016/J.ESWA.2021.114843

APÊNDICE A

Tabela 25 – Resultados e tempos computacionais das instâncias pequenas

	Solução 1			**Solução 2**			**Solução 3**		
Instância	Tempo (s)	*Tard.* (min)	S. Cmax (min)	Tempo (s)	*Tard.* (min)	S. Cmax (min)	Tempo (s)	*Tard.* (min)	S. Cmax (min)
P_1	36,02	1730	9580	35,61	1730	9583	36,12	1730	9577
P_2	35,71	2117	10402	35,91	2117	10308	36,29	2117	10410
P_3	22,12	3400	11314	21,87	3400	11310	23,62	3400	11310
P_4	24,61	2817	11676	24,58	2817	12033	24,81	2817	12033
P_5	27,19	7485	11298	25,89	7485	11298	30,34	7485	11298
P_6	21,06	1048	9418	20,69	1048	9224	21,28	1048	9207
P_7	25,37	719	11101	25,89	719	11101	26,56	719	11101
P_8	18,53	1824	10191	18,69	1787	10217	18,97	1787	10217

ANEXO A

The identical parallel machine scheduling problem with setups and additional resources

Ângelo Soares[1[0000-0003-0783-7295]], Ana Rita Ferreira[1[0000-0001-9215-7192]] and Manuel Pereira Lopes[1,2*[0000-0002-1146-7226]]

[1]ISEP, Polytechnic of Porto, R. Dr. António Bernardino de Almeida, 4249-015 Porto, Portugal

[2] INESC TEC, R. Dr. Roberto Frias 400, 4200-465 Porto, Portugal

*mpl@isep.ipp.pt

Abstract. This paper studies a real world dedicated parallel machine scheduling problem with sequence dependent setups, different machine release dates and additional resources (PMSR). To solve this problem, two previously proposed models have been adapted and a novel objective function, the minimisation of the sum of the machine completion times, is proposed to reflect the real conditions of the manufacturing environment that motivates this work. One model follows the strip-packing approach and the other is time-indexed. The solutions obtained show that the new objective function provides a compact production schedule that allows the simultaneous minimisation of machine idle times and setup times. In conclusion, this study provides valuable insights into the effectiveness of different models for solving PMSR problems in real-world contexts and gives directions for future research in this area using complementary approaches such as matheuristics.

Keywords: Parallel machines, Scheduling, Setups, Additional Resources

1. Introduction

The study of Parallel Machine scheduling problems (PM) has mostly relied on machines as the primary resource, neglecting the fact that real manufacturing settings involve other critical resources such as automated guided vehicles, machine operators, tools, pallets, dies, and industrial robots [1-3]. The PM with sequence dependent setup times (PMS) where additional processing and setup resources are required (PMSR) is more complex. It involves sequencing and timing decisions and balancing multiple performance metrics. This study focuses on the injection moulding department of an automotive company that produces plastic injection moulded parts for final assembly plants with a maximum response time of 48 hours. To solve this real problem, two existing models are adapted to the PMSR involving identical PM with different machine release dates, and a new objective function that minimises the sum of the completion times for each machine is proposed. This objective function provides a compact production schedule that allows the simultaneous minimisation of machine idle times and setup times for the specific case of dedicated PM with sequence dependent setup times. To the authors' knowledge, no existing research has explored the minimization of the sum of completion times for each machine as an objective function. This study provides valuable insights into the effectiveness of different models for solving this type of PMSR problem in real-world contexts.

2. Literature review

In recent years, several papers have addressed the more general case of PMSP with setups and resources. Akyol Ozer, et al. [4] studied the identical PMSP with sequence-dependent setup times, machine eligibility restrictions and multiple copies of shared resources. In this paper, the objective function was the minimization of the total weighted completion time. For small instances (8 jobs, 3 machines and 5 resources), a mixed integer linear programming model (MILP) obtained optimal solutions in almost all tests. For larger problems, a matheuristic consisting of a MILP and a Genetic Algorithm (GA) has been implemented, which provides solutions for up to 100 jobs, 8 machines and 50 resources. Bektur, et al. [5] presented a MILP and two metaheuristics, Tabu Search (TS) and Simulated Annealing (SA), for the case of a team as a setup resource and the objective of minimizing the total weighted tardiness. MILP only gives optimal solutions for 2 machines and 10 jobs with a time limit of 7200 s. Regarding the metaheuristics, for up to 10 machines and 100 jobs, TS outperformed SA with an average relative deviation of 5.58%. Pinheiro, et al. [6] propose a solution approach to the unrelated PMSP with family setups and resource constraints, to minimize the total tardiness of the jobs. The authors develop a MILP and two metaheuristics, SA and Iterated Greedy (IG). The MILP showed good performance for instances with up to 20 jobs and 6 machines, with a computational time limit of 3600 s. To address larger problems, a constructive heuristic was used to provide an initial solution, which was then improved by the metaheuristics. For instances with up to 50 jobs, the IG algorithm outperformed the SA algorithm, exhibiting lower relative deviation values. In fact, IG and SA algorithms have average relative deviations of 8.84% and 39.87%, respectively. The computational time required for the IG algorithm to provide a solution was also reported to be lower than the MILP model. Fanjul-Peyro [7] addressed the Unrelated PM with Setups and Resources (UPMSR), with three types of resources: specific for processing, specific for setup and shared. The authors presented two MILP models and a three-phase exact algorithm (TPhA) to improve the model's performance. Yepes-Borrero, et al. [8] implemented a GRASP algorithm with the objective of minimizing makespan and in [9] have also proposed a multi-objective approach to minimise both the makespan and the number of setup resources, building on the previous work [7, 8]. An adaptation of the Restarted Iterated Pareto Greedy (RIPG) algorithm is proposed that gives solutions for up to 250 jobs and 30 machines. Zhang, et al. [10] tackled the unrelated PMSP, with limited worker resources and learning effect. They proposed a Combinatorial Evolutionary Algorithm to minimize makespan and energy consumption. The algorithm can handle up to 400 jobs, 22 machines, and 10 resources. Lopez-Esteve, et al. [11] also developed GRASP algorithms to solve the unrelated PMSP with additional resources during processing and setups which gives solutions with a gap of 1.80% in relation to the Fanjul-Peyro [7]. Yunusoglu, et al. [12] proposed an exact model based on constraint programming (CP) to solve the multi-resource-constrained unrelated PMSP with sequence-dependent setup times. The objective was to minimize the makespan and release dates were incorporated into the problem as operational constraints to reflect real-world manufacturing environments. The computational results indicate that the proposed CP model outperforms the best solutions of previous works with an average gap of 15.52%. Finally, Afzalirad, et al. [13] inspired on the block erection scheduling problem in a shipyard, studied the unrelated PMSP with resource constrains, sequence-dependent setup times, different release dates, machine eligibility and precedence constraints, to minimize the makespan. The results indicate that in the case of small-scale problems, both methods are efficient and effective, whereas in large-scale problems, the AIS approach statistically outperforms the GA approach.

3. Problem Formulation

3.1. Problem description

We consider a parallel dedicated machine scheduling problem, where *n* jobs are processed on *m* machines, each job belonging to a family and processed on a single machine without preemption. A set of parts which use the same die is a family and a sequence dependent setup must be performed between jobs belonging to different families. Operators are required to process jobs and setup teams are required to perform the machine setups. Operators and setup teams are renewable and dynamic resources. The number of operators for each shift is known in advance, and the system must provide scheduling solutions at the start of each shift. The objective of the Injection Moulding Department is to meet the daily demand for parts within the constraints of capacity, and the availability of operators and setup teams, minimizing the maximum completion time for each machine.

3.2. Model formulation

In this section we formally introduce the dedicated parallel machine with setups and resources. In this paper we use small letters to refer to parameters and capital letters to refer to variables. The sets of variables and parameters used are the following:

- A set of *m* machines that can process the jobs, denoted as $M = \{1, \dots, m\}$, indexed by *i*.
- A set of *m*+*n* jobs to be processed, $N = \{1, \dots, m + n\}$, indexed by *j*, where the first *m* jobs represent the initial jobs on the respective machine number.
- p_j denotes the time required to process job *j*.
- s_{jk} denotes the time required to do the setup between jobs *j* and *k*.
- r_{max}^P are the maximum amount of processing resources. These are required specifically for processing jobs and cannot be used for setups. We denote these resources as P-resources.
- r_{max}^S are the maximum amount of machine setup resources. These are required specifically for machine setup and cannot be used for job processing. We denote these resources as S-resources.
- r_j^P are the resources of P-resources required to do processing of job *j*.
- r_j^S are the resources of S-resources required to do setup of job *j*.
- *M* is a sufficiently large constant.

As the machines are dedicated, the set of *N* jobs is divided into *m* subsets ($N_1, \dots, N_m$) so that the jobs in set N_i are only processed on machine *i*. Each machine has a dummy job (0) representing its start and end. Therefore, N_{i0}=$N_i \cup \{0\}$ represents the set of jobs which are to be processed on machine *i* plus the dummy job. Moreover, processing and setup times (p_j and s_{jk}), as well as the resources required for each operation (r_j^P and r_j^S) are deterministic. In this paper the UPMSR models based on strip-packing and indexed to time introduced by [7] are adapted to a real problem where the parallel machines are dedicated and have different release dates (the production system has an initial state), and a new objective function is proposed to minimize the sum of the machines completion times. The decision variables used are the following:

- $X_{ijk} = 1$ if job *k* is processed immediately after job *j* on machine *i*, zero otherwise.
- $Cmax_i$ is the completion time (minutes) of machine *i*.

- C_j is the completion time of job *j* (coordinate where processing of job *j* finishes in the time dimension) (minutes).
- B_j is the completion time of the setup before the processing of job *j* on the corresponding machine (coordinate where setup of job *j* finishes in the time dimension).
- $R_j^P \in [0, r_{max}^P]$ is the coordinate of processing of job *j* in the P-resources dimension (the specific resources employed in the processing of job *j*).
- $R_j^S \in [0, r_{max}^S]$ is the coordinate of setup of job *j* in the S-resources dimension (the specific resources employed in the setup before processing of job *j*).
- D_{jk}^P, E_{jk}^P, F_{jk}^S G_{jk}^S are binary auxiliary variables with value = 1 when the variable value studied in constraints (time or resources) of job *k* is higher than the variable value studied (time or resources) of job *j*. The letters in superscript are used to indicate the dimension where they are used.

The MILP strip-packing model is:

$$Min\ Z = \sum_{i \in M} Cmax_i \tag{1}$$

$$X_{i0i} = 1, i \in M \tag{2}$$

$$\sum_{j \in N_i, j \neq k} X_{ijk} = 1, i \in M, k \in N_i \backslash \{i\} \tag{3}$$

$$\sum_{j \in N_{i0} \backslash \{i\}, j \neq k} X_{ikj} = 1, i \in M, k \in N_i \tag{4}$$

$$C_k - C_j + M\left(1 - X_{ijk}\right) \geq s_{jk} + p_k, i \in M,\ j \in N_i, k \in N_i \backslash \{i\}, j \neq k \tag{5}$$

$$C_k \geq \sum_{j \in N_{i0}, j \neq k, s_{jk} > 0} X_{ijk}\, s_{jk} + p_k, k \in N_i \backslash \{i\}, i \in M \tag{6}$$

$$C_k \geq \sum_{j \in N_{i0}, j \neq k, s_{jk} > 0} X_{ijk}\, (s_{jk} + p_j) + p_k, k \in N_i \backslash \{i\}, i \in M \tag{6b}$$

$$B_k \geq \sum_{j \in N_{i0}, j \neq k, s_{jk} > 0} X_{ijk}\, (s_{jk} + p_j) + p_k, k \in N_i \backslash \{i\}, i \in M \tag{6c}$$

$$B_k \geq \sum_{j \in N_{i0}, j \neq k, s_{jk} > 0} X_{ijk}\, (s_{jk} + p_j), k \in N_i \backslash \{i\}, i \in M \tag{6d}$$

$$C_i \geq p_i, i \in M \tag{7}$$

$$Cmax_i \geq C_k, i \in M, k \in N_i \tag{8}$$

$$B_k - C_j + M\left(1 - X_{ijk}\right) \geq s_{jk},\ i\ \in M, j \in N_i, k\ \in N_i \backslash \{i\}, j \neq k, s_{jk} > 0 \tag{9}$$

$$B_k \leq C_k - p_k, k\ \in N \tag{10}$$

$$r_{max}^{P} \geq R_k^P, k \in N \quad (11)$$

$$R_k^P \geq r_k^p, k \in N \quad (12)$$

$$r_{max}^{S} \geq R_k^S, k \in N \quad (13)$$

$$R_k^S \geq r_k^S, k \in N \quad (14)$$

$$C_k - C_j + M\left(1 - D_{jk}^P\right) \geq p_k, i, q \in M, q \neq i, j \in N_i, k \in N_q \quad (15)$$

$$R_k^P - R_j^P + M\left(1 - E_{jk}^P\right) \geq r_k^P, i, q \in M, q \neq i, j \in N_i, k \in N_q \quad (16)$$

$$D_{jk}^P + D_{kj}^P + E_{jk}^P + E_{kj}^P \geq 1, i, q \in M, q \neq i, j \in N_i, k \in N_q, j \neq k \quad (17)$$

$$B_k - B_j + M\left(1 - F_{jk}^S\right) \geq \sum_{l \in N_{q0}, l \neq k, s_{lk} \geq 0} X_{qlk}\, s_{lk}, i, q \in M, q \neq i, j \in N_i, k \in N_q \quad (18)$$

$$R_k^S - R_j^S + M\left(1 - G_{jk}^S\right) \geq r_k^S, i, q \in M, q \neq i, j \in N_i, k \in N_q \quad (19)$$

$$F_{jk}^S + F_{kj}^S + G_{jk}^S + G_{kj}^S \geq 1, i, q \in M, q \neq i, j \in N_i, k \in N_q \quad (20)$$

$$X_{ijk} \in \{0,1\}, Y \geq 0, C_k \geq 0, B_k \geq 0\ j, k \in N_i, i \in M \quad (21)$$

The objective function (1) minimizes the sum of maximum completion time for each machine. Constraints (2) ensure that the initial jobs are the first to be processed immediately after the dummy jobs on each machine. The constraints (3) and (4) make sure that only one job is processed immediately before and only one job is processed immediately after each job. Constraints (5) break subtours and give the right scheduling order, because if *k* is processed after *j* on machine *i*, it forces finishing the processing of job *k* not earlier than s_{jk} + p_k. Constraints (6) set the lower limit of completion time of job *k* (C_k) to be greater or equal than the sum of its setup plus its processing time while constraints (7) do the same for the initial jobs. Constraints (8) establish the relationship between $Cmax_i$ of machine *i* and the completion times of jobs on that machine. Constraints (9) force that, if job j is the previous job done before job *k* on machine *i*, the finishing time of setup *k* must be later than the finishing time of processing of job *j* plus setup time of job *k*. Constraints (10) ensure that setup finishing time of job *k* is earlier than processing starting time of job *k*. Constraints (11-14) set the coordinate limits of specific resources used in processing (R_k^P) and used specifically in setups (R_k^S). Constraints (15) show that, if C_k is later than C_j, the difference between these completion times is greater than or equal to the processing time of job *k*. In (16) if R_k^P is greater than R_j^P, the difference between coordinates of these resource points is greater than or equal to the resources used in the job *k* processing. Constraints (17) force at least one of the binary variables to have the value of 1. Constraints (18) show that, if B_k is later than B_j, the difference between these setup finishing times is greater than or equal to the setup time of job *k*. In (19) if R_k^S is greater than R_j^S, the difference between coordinates of these resource points is greater than or equal to the resources used in the job *k* setup. Constraints (20) force at least one of the binary variables to have the value of 1. Finally, constraints (21) set the limits of the variables. In this model, in addition

to several simplifications of the original model, the objective function (1) and the constraints (2) are new to reflect the specific conditions of the real problem. The MILP model indexed to time is:

- $E_{ikt} = 1$ if *k* is being processed on machine *i* in time *t*, zero otherwise.
- $F_{ijkt} = 1$ if *k* is in setup after job *j* on machine *i* in time *t*, zero otherwise.
- t_{max} is the planning horizon.

E_{ikt} and F_{ijkt} are decision variables and t_{max} is a parameter. For this MILP we use Eqs. (1) – (10) and (21). The other constraints are:

$$\sum_{t \le t_{max}} E_{ikt} \ge p_k, i \in M, k \in N_i \tag{22}$$

$$C_k \ge tE_{ikt}, i \in M, k \in N_i, t \le t_{max} \tag{23}$$

$$C_k + 1 - p_k \le tE_{ikt} + M(1 - E_{ikt}), i \in M, k \in N_i, t \le t_{max} \tag{24}$$

$$\sum_{t \le t_{max}} F_{ijkt} \ge s_{jk} X_{ijk}, i \in M, j \in N_i, k \in N_i, j \ne k, s_{jk} > 0, k \ge m \tag{25}$$

$$B_k \ge \sum_{j \in N_i, j \ne k} tF_{ijkt}, i \in M, k \in N_i, t \le t_{max}, s_{jk} > 0, k \ge m \tag{26}$$

$$B_k + 1 - \sum_{j \in N_i, j \ne k, s_{jk} > 0} s_{jk} X_{ijk} \le \sum_{j \in N_i, j \ne k, s_{jk} > 0} tF_{ijkt} + M\left(1 - \sum_{j \in N_i, j \ne k, s_{jk} > 0} F_{ijkt}\right), i \in M, k \in N_i, t \le t_{max} \tag{27}$$

$$R^P_{max} \ge \sum_{i \in M, k \in N_i} r^P_k E_{ikt}, t \le t_{max} \tag{28}$$

$$R^S_{max} \ge \sum_{i \in M, j, k \in N_i, j \ne k, s_{jk} > 0} r^S_k F_{ijkt}, t \le t_{max} \tag{29}$$

Constraints (21) and (22) work together to ensure that the E_{ikt} variable is only 1 for the exact duration of time that job *k* is being processed on machine *i*. This prevents unnecessary idle time on the machine and ensures that each job is processed for the minimum amount of time necessary to complete it. Constraints (23) complements these constraints by requiring that when E_{ikt} is equal to 1, the current time *t* must be greater than or equal to the completion time of job *k* minus its processing time plus 1. Constraints (24) - (26) represent the same for setups, where the time considered is the setup time. Constraints (27) ensures that the sum of all resources used in processing at each instant *t* is less than or equal to the maximum number of resources available for processing. Constraints (28) ensures the same for setups.

4. Computational experiments

To generate the instances, the number of jobs, number of machines, processing and setup times, number of setup teams, number of operators available, and their use by jobs were defined based on the real data from the company, and the time is in minutes. The sets of problems, small, medium, and large, are presented in Table 1. For each of the sets, 10 instances were randomly generated using processing and setup times generated from distributions of *U*(20,240), and *U*(30,50), respectively, and the number of operators to support the processing of jobs on the machines generated from the distribution of *U*(1,4). Setup times must satisfy the triangular inequality. The setups require a setup team. For the time indexed model, t_{max} was set to 480 minutes (one shift).

Table 1. Sets of test instances

Instance	Machines	Jobs	Operators	Setup Teams
Small	4	12	8	1
Medium	8	24	16	2
Large	16	40	30	2

The CPLEX 22.1.1.0 solver was used, with programming done in Python using the Pulp library on Microsoft Visual Studio Code 2023 IDE. The computational tests were executed on an Intel Core i7-85654 CPU1.80GHz, with 8.00 GB of RAM. To ensure reasonable runtimes, a time limit of 1800 seconds was set. Throughout the computational study, the indicators analysed were the gap to the optimum solution, the percentage of optimal solutions obtained, and the computation time. The performance of two models, strip-packing and time-indexed, was analysed and the average results for each set of instances are presented in Table 2. For medium and large cases, only the results for the strip-packing model are presented, as the time-index model could not find a solution within the time limit. The strip-packing model gives good results for the small and medium instances, and for the large instances it could always find a feasible solution with and average gap of 14.09%. In contrast, the time-indexed model can only solve small problems.

Table 2. Comparison between strip-packing and time-indexed models

	Strip-packing			Time-indexed		
Instance	%Gap	%Opt	Time (s)	%Gap	%Opt	Time (s)
Small	0.00	100	0.91	5.6	60	1800
Medium	2.15	40	984.05	-	-	-
Large	14.09	0	1800	-	-	-

5. Conclusions and future research

This paper studies the special case of PMSR. To solve this problem, two previously proposed models have been adapted and a novel objective function, the minimisation of the sum of the machine completion times, is proposed to reflect the real conditions of the manufacturing environment that motivates this work. Overall, the strip-packing model outperforms the time-indexed model in solving this type of PMSR problem and shows a good potential for solving real-world problems and for future research in this area. The solutions obtained show that the new objective function provides a compact production schedule that allows the simultaneous minimisation of machine idle times and setup times for the PMSR with identical parallel machines with different release dates. To further enhance the performance of the strip-packing model, complementary approaches such as matheuristics can be explored. Matheuristics combine the strengths of mathematical

programming and metaheuristics to provide a hybrid optimization approach that can efficiently solve complex problems.

References

1. Blazewicz, J., Lenstra, J.K., Kan, A.R.: Scheduling subject to resource constraints: classification and complexity. Discrete applied mathematics 5, 11-24 (1983).
2. Błażewicz, J., Kubiak, W., Röck, H., Szwarcfiter, J.: Minimizing mean flow-time with parallel processors and resource constraints. Acta informatica 24, 513-524 (1987).
3. Ventura, J.A., Kim, D.: Parallel machine scheduling about an unrestricted due date and additional resource constraints. Iie Transactions 32, 147-153 (2000).
4. Akyol Ozer, E., Sarac, T.: MIP models and a matheuristic algorithm for an identical parallel machine scheduling problem under multiple copies of shared resources constraints. Top 27, 94-124 (2019).
5. Bektur, G., Saraç, T.: A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server. Computers & Operations Research 103, 46-63 (2019).
6. Pinheiro, J.C., Arroyo, J.E.C., Fialho, L.B.: Scheduling unrelated parallel machines with family setups and resource constraints to minimize total tardiness. In: Proceedings of the 2020 genetic and evolutionary computation conference companion, pp. 1409-1417. (2020).
7. Fanjul-Peyro, L.: Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. Expert Systems with Applications: X 5, 100022 (2020).
8. Yepes-Borrero, J.C., Villa, F., Perea, F., Caballero-Villalobos, J.P.: GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. Expert Systems with Applications 141, 112959 (2020).
9. Yepes-Borrero, J.C., Perea, F., Ruiz, R., Villa, F.: Bi-objective parallel machine scheduling with additional resources during setups. European Journal of Operational Research 292, 443-455 (2021).
10. Zhang, L., Deng, Q., Lin, R., Gong, G., Han, W.: A combinatorial evolutionary algorithm for unrelated parallel machine scheduling problem with sequence and machine-dependent setup times, limited worker resources and learning effect. Expert Systems with Applications 175, 114843 (2021).
11. Lopez-Esteve, A., Perea, F., Yepes-Borrero, J.C.: GRASP algorithms for the unrelated parallel machines scheduling problem with additional resources during processing and setups. International Journal of Production Research 1-17 (2022).
12. Yunusoglu, P., Topaloglu Yildiz, S.: Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. International Journal of Production Research 60, 2212-2229 (2022).
13. Afzalirad, M., Rezaeian, J.: Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. Computers & Industrial Engineering 98, 40-52 (2016).