UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

JOSÉ MAURÍCIO FERNANDES MEDEIROS

ABORDAGENS HEURÍSTICAS PARA O PROBLEMA DE ESCALONAMENTO DE LOTES EM MÁQUINAS PARALELAS COM PENALIDADES POR ATRASO

João Pessoa
2023

JOSÉ MAURÍCIO FERNANDES MEDEIROS

ABORDAGENS HEURÍSTICAS PARA O PROBLEMA DE ESCALONAMENTO DE LOTES EM MÁQUINAS PARALELAS COM PENALIDADES POR ATRASO

Dissertação submetida ao Programa de Pós-Graduação em Informática do Centro de Informática da Universidade Federal da Paraíba, como requisito parcial para obtenção do Grau de Mestre em Informática.

Orientador: Prof. Dr. Anand Subramanian

João Pessoa

2023

Catalogação na publicação
Seção de Catalogação e Classificação

M488a Medeiros, José Maurício Fernandes.
Abordagens heurísticas para o problema de escalonamento de lotes em máquinas paralelas com penalidades por atraso / José Maurício Fernandes Medeiros. - João Pessoa, 2023.
55 f. : il.

Orientação: Anand Subramanian.
Dissertação (Mestrado) - UFPB/CI.

1. Máquinas paralelas - Otimização combinatória. 2. Problema de escalonamento de lotes. 3. Meta-heurística. 4. Iterated local search. I. Subramanian, Anand. II. Título.

UFPB/BC CDU 004.388-048.34(043)

Elaborado por ANNA REGINA DA SILVA RIBEIRO - CRB-15/24

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de José Maurício Fernandes Medeiros, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 03 de julho de 2023.

Ao terceiro dia do mês de julho do ano de dois mil e vinte e três, às dezesseis horas, no Centro de Informática da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para julgar o trabalho do sr. José Maurício Fernandes Medeiros, vinculado a esta Universidade sob a matrícula nº 20211000490, candidato ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de pesquisa "Computação Distribuída", do Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores: Anand Subramanian, Orientador e Presidente da banca; Teobaldo Leite Bulhões Júnior (PPGI), Examinador Interno, e Rian Gabriel Santos Pinheiro (UFAL), Examinador Externo ao Programa. Dando início aos trabalhos, a Presidente da Banca cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado: "Abordagens Heurísticas para o Problema de Escalonamento de Lotes em Máquinas Paralelas com Penalidades por Atraso". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Fernando Menezes Matos, Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai assinada por mim e pelos membros da banca examinadora. João Pessoa, 03 de julho de 2023.

gov.br Documento assinado digitalmente
FERNANDO MENEZES MATOS
Data: 23/08/2023 17:23:32-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Fernando Menezes Matos

Prof. Anand Subramanian
Orientador (PPGI-UFPB)

Prof. Teobaldo Leite Bulhões Júnior
Examinador Interno (PPGI-UFPB)

Prof. Rian Gabriel Santos Pinheiro
Examinador Externo ao Programa (UFAL)

"*Remember to look up at the stars and not down at your feet. Try to make sense of what you see and wonder about what makes the universe exist. Be curious. And however difficult life may seem, there is always something you can do and succeed at. It matters that you don't just give up.*"

Stephen Hawking

AGRADECIMENTOS

No decorrer deste trabalho, incontáveis momentos de dificuldades foram enfrentados. Sem o apoio da minha família, orientador e colegas de trabalho, nada seria alcançado. Dessa forma, apresento meus sinceros agradecimentos:

- À minha esposa, Kalina, e aos meus filhos, Cecília, Olívia, Bernardo e Tomás, por serem minha fonte inesgotável de amor e motivação.
- À toda minha família, pelo encorajamento e apoio em todos os meus esforços.
- Ao meu grande amigo, professor e orientador, Anand Subramanian, por me convencer a me envolver novamente com a pesquisa científica, além da confiança depositada durante todo o programa.
- Aos meus colegas de trabalho por me incentivarem nessa empreitada e pelos sacrifícios dispensados.

RESUMO

Este trabalho aborda o problema de escalonamento de lotes em máquinas paralelas idênticas com penalidades por atraso, datas de liberação e famílias de tarefas. Nesse ambiente, lotes são constituídos por tarefas de uma mesma família e alocados a uma máquina para processamento. As tarefas agrupadas são processadas simultaneamente respeitando a data de liberação e tempo de processamento associados ao lote. O objetivo consiste em determinar um sequenciamento de lotes a serem processados em cada máquina a fim de minimizar a soma dos atrasos ponderados. Para resolver o problema, são propostos dois algoritmos meta-heurísticos empregando procedimento de busca local com múltiplas estruturas de vizinhança e de mecanismos de perturbação de soluções. Experimentos computacionais são realizados em instâncias conhecidas e os resultados são analisados e comparados com aqueles obtidos utilizando outros métodos reportados na literatura.

Palavras-chave: Otimização Combinatória. Problema de escalonamento de lotes. Máquinas paralelas. Meta-heurística. *Iterated local search*.

ABSTRACT

This work addresses the identical parallel batch machines scheduling problem subject to tardiness penalties, release dates, and job families. In this environment, jobs of the same family are partitioned into batches and each batch is assigned to a machine. The jobs are processed simultaneously and are subject to the release date and processing time of the batch to which they belong. The objective is to determine the sequence of batches to be processed on each machine in order to minimize the total weighted tardiness. To solve the problem, two metaheuristic algorithms are proposed, employing local search procedures with multiple neighborhood structures and perturbations mechanisms. Computational experiments are conducted on known instances, and the results are analyzed and compared with those obtained using other methods found in the literature.

Keywords: Combinatorial Optimization. Batch scheduling problem. Parallel machines. Metaheuristics. Iterated local search.

SUMÁRIO

LISTA DE TABELAS

GLOSSÁRIO

AG	Algoritmo Genético
AM	Algoritmo Memético
ATC	*Apparent Tardiness Cost*
BATC-II	*Batched Apparent Tardiness Cost II*
CF	Colônia de Formigas
GRASP	*Greedy Randomized Adaptive Search Procedure*
HGSADC	*Hybrid Genetic Search with Adaptative Diversity Control*
IG	*Iterated Greedy*
ILS	*Iterated Local Search*
PILS	*Population-Based Iterated Local Search*
LNS	*Large Neighborhood Search*
PEL	Problema de Escalonamento de Lotes
PLI	Programação Linear Inteira
PLIM	Programação Linear Inteira Mista
RVND	*Randomized Variable Neighborhood Descent*
SA	*Simulated Annealing*
TWD	*Time Window Decomposition*
VND	*Variable Neighborhood Descent*
VNS	*Variable Neighborhood Search*

Capítulo 1

Introdução

1.1 Definição do tema

O problema de escalonamento de lotes (PEL), também conhecido na literatura como *batch scheduling* (BS), é um problema de otimização combinatória, sendo uma generalização do clássico problema da mochila. Tem como finalidade agrupar tarefas em lotes e realizar seu processamento com uso dos recursos disponíveis. Essa designação deve atender às restrições estabelecidas e alcançar o objetivo do problema da melhor forma possível, ou seja, empregando de forma eficiente os recursos à disposição (Pinedo, 2018).

O PEL consiste, de forma geral, em definir quais tarefas constituirão cada lote, a alocação deste a uma máquina e seu momento de execução, estabelecendo, portanto, um cronograma de processamento para cada máquina. Esse processo decisório tem grande relevância no planejamento e controle da produção de uma organização, refletindo no aumento da capacidade através do melhor uso dos recursos.

Na literatura, há um vasto número de trabalhos acerca do PEL, muitos dos quais relacionados a aplicações práticas em setores da indústria. Por essa natureza, um abundante número de variantes emergem em razão das diferentes especificidades de cada processo produtivo. Há problemas que empregam somente uma máquina, enquanto outros fazem uso de múltiplas em paralelo, podendo, inclusive, ser idênticas ou não.

Tarefas podem ser agrupadas em famílias, o que permite um uso mais proveitoso dos recursos, seja pela formação de *batches* ou pelo seu processamento serial. Há, ainda, atributos inerentes a essas tarefas que podem ser considerados, como tamanho, penalidade por atraso, tempo de processamento, datas de liberação e de entrega, entre outros. Além disso, os objetivos do problema podem variar conforme a necessidade da organização. Nesse sentido, há estudos que se concentram em minimizar o tempo total de processamento, a soma dos atrasos ponderados, entre outras possibilidades. Esses são apenas alguns exemplos das diversas alternativas de abordagem do problema.

Problemas dessa natureza são modelados com uso de uma função objetivo e de um conjunto de restrições às quais a solução deve atender. Deseja-se alcançar o resultado ótimo, ou seja, o melhor entre todos os possíveis. No entanto, a depender da complexidade do modelo e do tamanho das instâncias a serem processadas, o problema pode se tornar computacionalmente intratável. Nestes casos, importa-se em solucionar o problema com qualidade em um tempo computacional razoável.

De acordo Lawler (1977), o problema que envolve o processamento de um conjunto tarefas em uma única máquina, definido como $1|\sum w_j T_j$ segundo notação proposta por Graham et al. (1979), é *NP-difícil*. Como a variante abordada neste trabalho generaliza o problema citado, pode-se afirmar que ela é, invariavelmente, *NP-difícil*.

Este projeto aborda o problema de escalonamento de lotes com máquinas paralelas idênticas, penalidades por atraso, datas de liberação e famílias de tarefa. Conforme a notação convencionada, este problema é definido como $P_m|r_j, batch, incompatible|\sum w_j T_j$.

1.2 Motivação

A abundância de estudos científicos relacionadas ao PEL demonstra sua relevância para a academia e para os setores produtivos, onde são encontradas aplicações práticas nas indústrias de semicondutores, sapatos, aeronaves, aço, móveis, recipientes de vidro, entre outras (Mathirajan et al., 2010). Esse problema tem um papel essencial nos processos organizacionais, proporcionando maiores níveis de reutilização de recursos, atendimento de datas de entrega e produção. Em outras palavras, o PEL se empenha

em promover um melhor uso dos recursos disponíveis aplicados a um determinado processo, ou seja, maximizar sua utilização em benefício da produtividade.

Há um extenso número de trabalhos relacionados à indústria de semicondutores (Blazewicz et al., 2019; Fowler e Mönch, 2022; Vimala Rani e Mathirajan, 2022; Fang et al., 2023). Nessa área, fábricas produzem circuitos integrados empregando *wafers*, discos finos de material semicondutor, em geral, a base de silício, em um processo demorado e complexo (Fowler e Mönch, 2022). As atividades envolvidas são classificadas como *front-end*, onde ocorre, efetivamente, a produção dos *chips*, e *back-end*, onde se realizam as atividades de montagem de componentes e testes. Em todas essas fases de produção há aplicabilidade de técnicas relacionadas ao PEL. No processo conhecido como *burn-in*, os *chips* montados são submetidos a um teste de calor para identificar aqueles mal fabricados. A depender do tamanho, cada item é posicionado em uma ou mais placas. Estas, por sua vez, são agrupadas em lotes e colocadas em um forno. Como se trata de uma das últimas etapas do processo, o resultado do escalonamento é crítico para determinar entregas dentro do prazo (Lee et al., 1992).

Na indústria aeronáutica, conhecidamente competitiva, Van der Zee et al. (1997) desenvolveram uma estratégia de controle da produção de partes sintéticas de aeronaves. Nesse ambiente, partes de determinado tipo são agrupadas em lotes e processadas em fornos industriais sujeitas a temperaturas, pressões e tempos de processamento específicos. O tamanho do lote é limitado pelo forno e o processamento dos seus itens ocorre de forma uniforme e sem interrupções. Com base nesses requisitos, busca-se otimizar o custo logístico e o nível de serviço.

O ambiente hospitalar também oferece aplicações para o problema de escalonamento. Nesse sentido, Ozturk et al. (2012) estudaram o processo de lavagem de itens médicos no serviço de esterilização de um hospital. Entre os itens estão instrumentos, aparelhos ou qualquer artigo médico empregado no tratamento de pacientes. Pode-se destacar aqueles relacionados a procedimentos cirúrgicos. Após o uso, esses componentes são transferidos para o serviço de esterilização, onde são empacotados individualmente ou em grupos, compondo lotes. Finalmente, cada lote é processado por uma máquina de lavar automática. Em virtude da presença de inúmeros itens reusáveis para uma quantidade limitada de máquinas, deve-se promover da melhor forma possível sua

utilização para atender as necessidades da organização.

Atividades de planejamento nesses cenários são complexas e envolvem decisões associadas a diversas variáveis e restrições, o que torna o escalonamento de tarefas um trabalho desafiador. Dessa forma, e considerando a natureza *NP-difícil* dessa classe de problema, resta claro a necessidade de estudos para o desenvolvimento métodos computacionalmente eficientes para melhorar a qualidade das soluções dos cenários apresentados.

1.3 Objetivos

1.3.1 Objetivo geral

Propor algoritmos meta-heurísticos eficientes para resolver o problema $P_m|r_j, batch, incompatible|\sum w_j T_j$.

1.3.2 Objetivos específicos

Os objetivos específicos para atender a proposta deste projeto estão elencados a seguir.

- Desenvolver um algoritmo para geração de soluções iniciais.
- Implementar um procedimento de busca local utilizando múltiplas estruturas de vizinhança.
- Implementar mecanismos de pertubação de soluções.
- Testar os algoritmos propostos em instâncias disponíveis na literatura.
- Avaliar o desempenho, comparando os resultados obtidos com aqueles encontrados por outros métodos apresentados na literatura.

1.4 Estrutura da dissertação

O restante deste trabalho está organizado da forma como apresentado abaixo.

- O Capítulo 2 apresenta uma revisão dos trabalhos relacionados.
- O Capítulo 3 descreve o problema e uma formulação matemática.
- O Capítulo 4 explica em detalhes os algoritmos propostos.
- O Capítulo 5 apresenta os resultados dos experimentos computacionais.
- O Capítulo 6 analisa o trabalho desenvolvido e sugere possibilidades de extensão da pesquisa.

Capítulo 2

Revisão da literatura

Este capítulo oferece uma visão geral dos principais atributos relacionados ao problema de escalonamento de tarefas, do qual o PEL deriva, descrevendo propriedades, configuração das máquinas, restrições e objetivos. Em sequência, é apresentada uma revisão da literatura abordando estudos de variantes idênticas ou conectadas àquela discutida neste trabalho.

2.1 Atributos dos problemas de escalonamento

As variantes do problema de escalonamento de tarefas são definidas em função da combinação dos diferentes atributos, restrições e objetivos envolvidos. Há processos que consideram apenas uma máquina no processamento das tarefas, enquanto outros utilizam múltiplas, podendo estar submetidas a tempos de configuração. As restrições às quais o problema está sujeito podem alterar severamente a sua complexidade, como a formação de lotes, presença de famílias de tarefas, datas de liberação, *deadlines*, entre outras. Portanto, o conhecimento desses elementos é fundamental para a contextualização e entendimento dos estudos relacionados a essa área de pesquisa.

Nessa perspectiva, a notação elaborada por Graham et al. (1979) simplifica a identificação do tipo de problema, descrevendo-o com base em três componentes: $\alpha|\beta|\gamma$. O primeiro item α oferece uma visão da arquitetura das máquinas. O componente β

descreve as restrições consideradas e γ o objetivo a ser alcançado. Os principais elementos associados a esses componentes são listados nesta seção, enquanto uma relação mais abrangente pode ser encontrada nos trabalhos de Pinedo (2018) e Fowler e Mönch (2022).

2.1.1 Tarefas

As tarefas possuem um conjunto de propriedades que as caracterizam e são empregadas como parâmetros de entrada na resolução do problema. Abrangem dados de processamento e limites que devem ser atendidos.

- Tamanho (s_j): cada tarefa possui um determinado tamanho. Para serem processadas em uma máquina, esse valor deve ser menor ou igual à sua capacidade. Quando os tamanhos das tarefas não são uniformes, esse elemento é apresentado no componente β.

- Tempo de processamento (p_j): representa o tempo de processamento da tarefa j. Todavia, em determinados cenários, esse tempo pode variar conforme a máquina (p_{ij}).

- Data de liberação (r_j): se refere ao momento em que a tarefa está disponível no sistema. Dessa forma, a tarefa somente pode ser processada a partir dessa data. Nem todos os problemas consideram datas de liberação, portanto, quando incluídas, essa informação deve ser apresentada no componente β.

- Data de entrega (d_j): indica a data em que a tarefa deve ser entregue. Portanto, o seu processamento deve ser concluído até essa data, caso contrário, uma penalidade poderá ser aplicada.

- Penalidade (w_j): a penalidade se refere ao peso da tarefa em relação às demais. Em síntese, equivale a sua prioridade no sistema. Essa propriedade costuma estar presente no componente γ como base de cálculo do objetivo.

- *Deadline* ($\bar{d_j}$): corresponde à data limite para conclusão de uma tarefa. Diferentemente de d_j, o *deadline* deve, obrigatoriamente, ser respeitado. Essa restrição, quando presente, está disposta no componente β.

2.1.2 Máquinas

O primeiro componente da notação (α) se refere ao ambiente de processamento das tarefas. Essa informação reflete a organização das máquinas e aponta o modo de processamento dos itens. Assim como as tarefas, as máquinas possuem características próprias como capacidade máxima, velocidade de processamento e tempo de configuração. Os ambientes apresentados a seguir são aqueles com configuração mais próxima do problema abordado neste trabalho:

- Máquina única (1): cenário mais simples, onde há somente uma máquina disponível para processamento.
- Máquinas paralelas idênticas (P_m): neste ambiente, há múltiplas máquinas idênticas em paralelo e o tempo para processar cada tarefa p_j é o mesmo nas diferentes máquinas.
- Máquinas paralelas uniformes (Q_m): as múltiplas máquinas em paralelo desse sistema se diferenciam em razão da velocidade de processamento. Cada máquina possui uma determinada velocidade v_i e o tempo de conclusão da tarefa j é dado por p_j/v_i.
- Máquinas paralelas não relacionadas (R_m): nesse cenário, as velocidades de processamento das máquinas podem variar em razão das tarefas (v_{ij}). Esse ambiente generaliza os anteriores e a velocidade para processar a tarefa j na máquina i é dado por p_j/v_{ij}.

2.1.3 Restrições

As restrições às quais o problema está submetido estão dispostas no componente β da notação. Esse campo, diferentemente dos demais, aceita a presença de vários elementos. Aquelas comumente empregadas estão listadas abaixo:

- Formação de lotes (*batch*): aponta a necessidade de agrupamento de tarefas em lotes para processamento. Neste caso, o tamanho do lote, ou seja, o somatório

dos tamanhos de cada uma das tarefas que o compõe, deve ser menor que a capacidade da máquina.

- Família de tarefas ($fmls$): cada tarefa do problema está vinculada a uma determinada família. Nesse cenário, apenas itens de uma mesma família podem formar um lote, ou seja, as tarefas das diferentes famílias são incompatíveis entre si. Se não houver essa indicação na notação do problema, quaisquer itens podem participar da composição de um lote. Essa característica também pode ser denotada pelo termo *incompatible*.

- Precedência ($prec$): afirma que há itens no problema que, para serem executados, necessitam de processamento anterior de uma ou mais tarefas.

- *Breakdowns* ($brkdwn$): quando presente na notação, implica que nem todas as máquinas estarão disponíveis para processamento de forma contínua, podendo existir, por exemplo, períodos de manutenção.

- Elegibilidade de máquina (M_j): essa restrição evidencia que o processamento da tarefa j somente pode ser realizado por um conjunto de máquinas M_j. Quando ausente, os itens podem ser processados por qualquer recurso.

- Tempo de configuração (s_{jk}): indica que o problema considera tarefas que necessitam de configuração de máquina para poderem ser processadas. No caso, um tempo de ajuste é necessário entre as execuções das tarefas j e k. Se esse tempo de preparação variar em razão das máquinas, acrescenta-se o índice correspondente s_{ijk}.

2.1.4 Objetivos

O objetivo de cada problema é apresentado no componente γ. Nesse sentido, a função objetivo correspondente é responsável pela medida de qualidade dos resultados, sendo calculada em função do tempo de conclusão das tarefas C_j (Pinedo, 2018). Os elementos a seguir compreendem alguns exemplos dos vários encontrados na literatura:

- Tempo total de processamento ou *makespan* (C_{max}): nessa configuração, busca-se reduzir o tempo total de processamento das tarefas. O valor de C_{max} equivale ao tempo de conclusão C_j da última tarefa processada no sistema.

- Máximo atraso ou *maximum lateness* (L_{max}): nesse caso, a resolução do problema almeja diminuir o maior atraso de uma tarefa no sistema, onde $L_j = C_j - d_j$.

- Soma dos tempos de conclusão ponderados ou *total weighted completion time* ($\sum w_j C_j$): o objetivo é minimizar a soma dos tempos de conclusão das tarefas de forma ponderada.

- Soma dos atrasos ponderados ou *total weighted tardiness* ($\sum w_j T_j$): nessa configuração, deseja-se diminuir a soma dos atrasos ponderados, onde $T_j = \max(C_j - d_j, 0)$.

2.2 Trabalhos relacionados

Há uma ampla quantidade de estudos relacionados ao problema de escalonamento. Trabalhos que realizam revisões sistemáticas da literatura oferecem uma importante base para o estudo das distintas classes do problema. Nesse sentido, Kramer e Subramanian (2019) apresentaram uma extensa bibliografia sobre problemas associados ao escalonamento de tarefas. Os trabalhos analisados foram reunidos de acordo com suas características e os métodos de resolução apontados, oferecendo uma valiosa fonte técnica de pesquisa. Fowler e Mönch (2022) apresentaram uma destacada compilação de estudos referentes ao problema de escalonamento de lotes. Os autores abordaram diversas variantes, técnicas de solução e direções para pesquisas futuras. Finalmente, Vimala Rani e Mathirajan (2022) forneceram uma revisão sistemática de pesquisas associadas com a indústria de semicondutores. Foram tratadas, especificamente, variantes relativas a escalonamentos determinísticos, ou seja, que não sofrem interferência de eventos estocásticos.

Com base no elevado número de pesquisas associadas ao problema de escalonamento, foram consideradas neste estudo somente aquelas onde o problema propõe a

formação de lotes (*batch*) buscando minimizar soma dos atrasos ponderados ($\sum w_j p_j$) em um ambiente com múltiplas máquinas em paralelo.

Dentro desse escopo, Balasubramanian et al. (2004) trataram variante similar ao deste trabalho, mas desconsiderando datas de liberação, e propuseram duas soluções baseadas no Algoritmo Genético (AG). A primeira versão, inicialmente, constitui lotes e, posteriormente, os atribui às máquinas utilizando uma estratégia amparada no AG. Ao final, ocorre o sequenciamento desses lotes nas referidas máquinas. A segunda versão, primeiramente, designa tarefas para máquinas empregando o AG. Logo após, há a formação dos lotes para, então, ocorrer o sequenciamento. De forma similar a esse estudo, mas considerando datas de liberação, Mönch et al. (2005) apresentaram uma abordagem semelhante, empregando dois métodos baseados no AG seguindo, cada um, a mesma sequência de procedimentos mencionados anteriormente. Mönch et al. (2006) desenvolveram uma nova abordagem para tratar a mesmo problema. Foi proposta uma heurística baseada no *Apparent Tardiness Cost* (ATC), regra para despacho de tarefas introduzida por Vepsalainen e Morton (1987), onde a qualidade das soluções produzidas depende, diretamente, do valor atribuído ao parâmetro *look-ahead*. Deste modo, foram implementadas técnicas de inteligência artificial para uma calibração mais eficiente desse parâmetro a fim de alcançar melhores resultados.

Em pesquisa realizada por Klemmt et al. (2009), que trataram problema similar ao deste trabalho, mas empregando máquinas não relacionadas, foram propostas duas abordagens de solução, a primeira com uso de um método híbrido, considerando o uso do *Time Window Decomposition* (TWD) em conjunto com uma resolução baseada em um modelo de Programação Linear Inteira Mista (PLIM), e a segunda empregando a técnica *Variable Neighbourhood Search* (VNS). A proposta que utiliza o VNS obteve melhores resultados em termos de qualidade da solução e tempo de processamento.

Em problema semelhante ao tratado neste trabalho, mas desconsiderando datas de liberação, Venkataramana e Raghavan (2010) propuseram um algoritmo baseado na meta-heurística Colônia de Formigas (CF). Nessa abordagem, a regra ATC é aplicada na elaboração dos lotes e o algoritmo CF é responsável por atribuí-los sequencialmente às máquinas. O mesmo problema foi analisado por Almeder e Mönch (2011), que sugeriram dois modelos de resolução com uso de meta-heurísticas, o primeiro com base

na CF e o segundo com uso do VNS. Em seus experimentos, identificaram a superioridade da proposta com VNS. Ainda, a abordagem com CF obteve resultados melhores que aqueles produzidos pelos métodos propostos por Balasubramanian et al. (2004). O problema em questão também foi investigado por Lausch e Mönch (2016), que descreveram abordagens baseadas em AG, CF e *Large Neighbourhood Search* (LNS) para solucioná-lo. Os resultados apresentados pelo método LNS demonstraram ser superiores às das demais propostas e comparáveis com aqueles encontrados pela estratégia com VNS de Almeder e Mönch (2011).

Problema idêntico ao deste trabalho foi investigado por Chiang et al. (2010), que propuseram uma solução baseada no Algoritmo Memético (AM). Esta estratégia apresentou melhores resultados em termos de qualidade e tempo de processamento em relação aos achados de Mönch et al. (2005). Bilyk et al. (2014) estudaram problema semelhante, incluindo a possibilidade de precedência entre tarefas. Os autores desenvolveram duas abordagens, uma baseada no VNS e outra no *Greedy Randomized Adaptive Search Procedure* (GRASP), sendo a primeira aquela que apresentou melhores resultados. Finalmente, em estudo comparativo entre a abordagem vencedora e o AM proposto por Chiang et al. (2010), o VNS também esteve a frente. Em interessante pesquisa, Fidelis e Arroyo (2017) apresentaram algoritmos baseados nas meta-heurísticas *Iterated Greedy* (IG) e *Simulated Annealing* (SA) para solucionar problema idêntico ao deste estudo. Os autores, ainda, reimplementaram a solução baseada no AM de Chiang et al. (2010) e aquela baseada no VNS de Bilyk et al. (2014) segundo as respectivas propostas. Quando comparados com instâncias pequenas, os algoritmos IG, SA e VNS se comportaram de forma equivalente entre si e melhor em relação ao AM. Entretanto, quando aplicados a instâncias grandes, de forma geral, o algoritmo SA se sobressaiu em relação aos demais, apresentando os melhores resultados. A proposta com uso do IG foi superior ao VNS e o AM obteve o pior resultado.

Capítulo 3

Definição do problema e formulação matemática

Este capítulo apresenta uma definição e, adicionalmente, uma formulação matemática para o problema abordado neste trabalho.

3.1 Definição do problema

Seja o conjunto de tarefas $J = \{1, \ldots, n\}$ a serem processadas por múltiplas máquinas idênticas em paralelo $M = \{1, \ldots, m\}$. Cada tarefa $j \in J$ possui os seguintes atributos: família à qual pertence f_j, tempo de processamento p_j, tamanho s_j, data de liberação r_j, que se refere à data a partir da qual o item pode ser processado, data de entrega d_j e penalidade por atraso w_j. Cada tarefa j pertence a uma única família e lotes não podem conter itens de famílias diferentes. Ainda, a quantidade de itens em um lote é restringido pela capacidade das máquinas. Como são idênticas, assumem, cada uma, o valor Q. O somatório dos tamanhos de todas as tarefas de um lote é limitado por esse parâmetro.

O conjunto de lotes de um determinado problema somente pode ser quantificado quando todas as tarefas estiverem devidamente alocadas. Durante o processamento do lote pela máquina, todos os itens que o compõem são processados de forma simultânea,

ou seja, iniciam e terminam ao mesmo tempo, não sendo permitida interrupções. O tempo de processamento do lote B_{bk}, onde b indica sua posição na máquina k, se refere ao maior tempo de execução dentre todos os itens que o constitui, dessa forma $P_{bk} = \max\{p_j | j \in B_{bk}\}$. Cada lote somente pode ter seu processamento iniciado após o cumprimento das datas de liberação de todas as suas tarefas. Assim, considera-se a data de liberação do lote sendo $R_{bk} = \max\{r_j | j \in B_{bk}\}$. Importante salientar que o momento de liberação citado não se confunde com o início, de fato, do seu processamento S_{bk}, pois o lote somente poderá ser processado na máquina após o término do processamento anterior, ou seja, não há possibilidade de ocorrer execuções em paralelo na mesma máquina. Portanto, o tempo de conclusão do processamento do lote B_{bk} é dado por $C_{bk} = S_{bk} + P_{bk}$, onde $S_{bk} = \max\{C_{(b-1)k}, R_{bk}\}$.

O tempo de conclusão de uma tarefa C_j é determinado pelo momento de finalização do processamento do lote ao qual pertence C_{bk} e o atraso referente a cada item j é dado por $T_j = \max\{0, C_j - d_j\}$. Finalmente, o objetivo do problema é designar os lotes de forma sequenciada para cada máquina a fim de minimizar a soma dos atrasos ponderados das tarefas:

$$min \sum_{j=1}^{n} w_j T_j. \tag{3.1}$$

3.2 Formulação matemática

A formulação matemática do problema tratado neste trabalho foi elaborada com base nos modelos apresentados em Klemmt et al. (2009) e Zhang et al. (2023).

Tabela 3.1: Índices usados no modelo matemático.

Índice	**Referência**	**Valores**
j	Tarefas	$j \in \{1, \ldots, n\}$
k	Máquinas	$k \in \{1, \ldots, m\}$
b	Lotes de uma máquina	$b \in \{1, \ldots, b_{max}\}$.

Tabela 3.2: Variáveis do modelo matemático.

Variável	**Descrição**
x_{jbk}	Indica se a tarefa j está vinculada ao lote b da máquina k Assume 1, caso positivo, ou 0, caso contrário
S_{bk}	Início do processamento do lote b na máquina k
C_{bk}	Tempo de conclusão do lote b na máquina k
C_j	Tempo de conclusão da tarefa j
T_j	Atraso da tarefa j.

As Tabelas 3.1 e 3.2 apresentam os índices e as variáveis usados no modelo, respectivamente. A variável x_{jbk} indica se a tarefa j está vinculada ao lote b da máquina k e s_{bk} representa o início do processamento de b na máquina k. O modelo de programação linear inteira pode ser escrito da seguinte forma:

$$\min \sum_{j=1}^{n} w_j T_j \tag{3.2}$$

sujeito a:

$$\sum_{k=1}^{m} \sum_{b=1}^{b_{max}} x_{jbk} = 1, \qquad j \in J \tag{3.3}$$

$$\sum_{j=1}^{n} s_j x_{jbk} \leq Q, \qquad b \in B, k \in M \tag{3.4}$$

$$x_{ibk} + x_{jbk} \leq 1, \qquad i, j \in J, b \in B, k \in M; f_i \neq f_j \tag{3.5}$$

$$S_{bk} \geq r_j x_{jbk}, \qquad j \in J, b \in B, k \in M \tag{3.6}$$

$$S_{bk} \geq C_{(b-1)k}, \qquad b \in B \setminus \{B_{1k}\}, k \in M \tag{3.7}$$

$$\hat{M}(1 - x_{jbk}) + C_{bk} \geq S_{bk} + p_j, \qquad j \in J, b \in B, k \in M \tag{3.8}$$

$$C_j \geq C_{bk} x_{jbk}, \qquad j \in J, b \in B, k \in M \tag{3.9}$$

$$T_j \geq C_j - d_j, \qquad j \in J \tag{3.10}$$

$$x_{jbk} \in \{0, 1\}, \qquad j \in J, b \in B, k \in M \tag{3.11}$$

$$S_{bk}, C_{bk}, C_j, T_j \geq 0, \qquad j \in J, b \in B, k \in M. \tag{3.12}$$

A função objetivo (3.2) minimiza a soma dos atrasos ponderados de todas as tarefas.

As restrições (3.3) garantem que cada tarefa esteja alocada a somente um lote de uma máquina. As restrições (3.4) determinam que a soma dos tamanhos dos itens de um lote não seja maior que a capacidade máxima da máquina. As restrições (3.5) asseguram que apenas tarefas de uma mesma família estarão presentes em cada lote. As restrições (3.6) certificam que um lote só poderá ser processado após a maior data de liberação prevista das tarefas que o compõe. Em complemento, as restrições (3.7) indicam que esse processamento somente pode ser iniciado após a conclusão do lote anterior, portanto, não há possibilidade de ocorrer execuções em paralelo na mesma máquina. As restrições (3.8) delimitam a conclusão do lote em função do momento do seu início e do maior tempo de processamento das suas tarefas. Nesse caso, $\hat{M}$ é uma constante numérica de valor positivo suficientemente grande. Em sequência, as restrições (3.9) sustentam que a conclusão de cada tarefa esteja vinculada com o tempo de conclusão do lote ao qual pertence. As restrições (3.10) referem-se ao cálculo do atraso de cada tarefa. Finalmente, as restrições (3.11) e (3.12) determinam os valores permitidos pelas variáveis.

Capítulo 4

Algoritmos propostos

Este capítulo descreve os algoritmos implementados para solucionar o problema abordado neste trabalho. As soluções desenvolvidas se basearam em abordagens conhecidas na literatura na resolução de problemas de otimização, mas ainda não aplicadas à classe do PEL em estudo. O primeiro método se refere a uma adaptação daquele apresentado por Queiroga et al. (2021), que aplica a meta-heurística *Iterated Local Search* (ILS) a um problema que considera apenas uma máquina. O segundo método emprega uma abordagem populacional baseada no próprio ILS, conhecida como *Population-based Iterated Local Search* (PILS). Essa estratégia adotou alguns dos mecanismos de controle de população presentes na proposta de Vidal et al. (2012), que desenvolveram uma solução baseada no AG na resolução de variantes do Problema de Roteamento de Veículos (PRV). Os autores reportaram resultados bastante satisfatórios, sendo seus trabalhos, portanto, objetos de análise.

4.1 Algoritmo $\text{ILS}_{\text{batch-pm}}$

Esta proposta, uma adaptação do método apresentado por Queiroga et al. (2021) para o problema com múltiplas máquinas em paralelo, aplica a meta-heurística ILS em conjunto com procedimentos de busca local com múltiplas estruturas de vizinhanças e com mecanismos de pertubação de soluções. As seções a seguir especificam esses

componentes.

4.1.1 Meta-heurística ILS

O ILS é uma meta-heurística que tem como essência a construção e o aprimoramento de soluções através da combinação de estágios de perturbação, também conhecido como diversificação, e intensificação, aplicando procedimento de busca local. A perturbação insere alterações aleatórias em determinada solução, permitindo que a busca local, com uso das estruturas de vizinhança, possa se aprofundar na pesquisa por diferentes soluções em um novo espaço de busca. Essa estratégia oferece a possibilidade do algoritmo escapar de ótimos locais e convergir para resultados melhores. Apesar de sua simplicidade, o ILS tem sido empregado com sucesso em inúmeros problemas de otimização, produzindo resultados robustos e altamente eficazes (Lourenço et al., 2019).

As linhas gerais do funcionamento do ILS estão representadas no Algoritmo 1. Como observado, a execução começa pela construção de uma solução inicial (linha 1). Em seguida, iterativamente, uma nova solução é criada com base naquela que apresentou o melhor resultado até o momento. Em sequência, ocorre o processamento da perturbação e da busca local (linhas 4 e 5, respectivamente). Posteriormente, caso essa solução atenda ao critério de aceitação, ela será aproveitada como base para as próximas iterações (linha 6). Finalmente, quando a condição de saída da iteração for satisfeita, o processamento é encerrado.

Algoritmo 1 ILS

1: $s^* =$ `solucaoInicial`()
2: **while** condição de parada não atendido **do**
3: $s' = s^*$
4: $s' =$ `perturbar`(s')
5: $s' =$ `buscaLocal`(s')
6: $s^* =$ `criterioAceitacao`(s^*, s')
7: **return** s^*

4.1.2 Funcionamento do $ILS_{batch\text{-}pm}$

Como mencionado na introdução desta seção, este algoritmo tem sua estrutura baseada na solução elaborada por Queiroga et al. (2021). Apesar da variante estudada neste trabalho considerar múltiplas máquinas em paralelo, a estrutura em alto nível do algoritmo e os parâmetros considerados permanecem similares. As diferenças se concentram no procedimento adotado para construção da solução inicial e no comportamento de certas estruturas de vizinhança.

Esta proposta tem seu plano de execução apresentado no Algoritmo 2. O método considera dois parâmetros de entrada: I_{ils}, que aponta a quantidade de iterações possíveis no laço interno (linha 4) sem que haja aprimoramento do resultado, e I_{mp}, que representa o máximo de movimentos de perturbação a serem aplicados à solução na iteração. O conceito do algoritmo proposto se baseia naquele apresentado no Algoritmo 1, no entanto, permitindo reinícios quando o limite de I_{ils} iterações sem melhora é atingido.

Algoritmo 2 $\text{ILS}_{\text{batch-pm}}$

input: I_{ils}, I_{mp}
1: $f^* = \infty$
2: **while** condição de parada não atendido **do**
3: $s = \texttt{solucaoInicial}()$
4: **for** $iter = 1$ to I_{ils} **do**
5: $s' = s$
6: $s' = \texttt{perturbar}(s', I_{mp})$
7: $s' = \texttt{buscaLocal}(s')$
8: **if** $f(s') < f(s)$ **then**
9: $s = s'$
10: $iter = 0$
11: **if** $f(s) < f^*$ **then**
12: $s^* = s$
13: $f^* = f(s)$
14: **return** s^*

4.1.3 Solução inicial

A construção de soluções iniciais (linha 3) adota a heurística *Time Window Decomposition* (TWD) em conjunto com as regras *Apparent Tardiness Cost* (ATC) (4.1), sugerida por Vepsalainen e Morton (1987), e *Batched Apparent Tardiness Cost II* (BATC-II) (4.2), formulada por Mönch et al. (2005). A combinação dessas regras, proposta por Mönch et al. (2005), resulta em soluções razoáveis com pouco tempo de processamento.

$$I_{ATC,j}(t) = \frac{w_j}{p_j} \exp\left(-\frac{(d_j - p_j - t + (r_j - t)^+)^+}{\kappa \overline{p}}\right) \tag{4.1}$$

$$I_{BATC-II,b}(t) = \frac{n_b}{B} \sum_{j \in J(b)} \frac{w_j}{p_j} \exp\left(-\frac{(d_j - p_j - t + (r_b - t)^+)^+}{\kappa \overline{p}}\right) \tag{4.2}$$

O Algoritmo 3 apresenta o funcionamento geral do TWD. Em resumo, a heurística, de forma iterativa, aplica a regra ATC para estabelecer índices de priorização das tarefas não alocadas. Na sequência, as tarefas que possuem alta prioridade e data de liberação numa determinada janela de tempo são agrupadas e combinadas entre si para formar lotes candidatos. Cada novo lote é analisado com base na regra BATC-II. Aquele com melhor índice é selecionado e adicionado à solução. Esse processo é repetido até que todas as tarefas sejam alocadas. Mais detalhes sobre essa técnica e os parâmetros utilizados podem ser encontrados em Mönch et al. (2005).

Como apresentado no Algoritmo 3, o TWD utiliza os parâmetros $thres$ e κ (*look-ahead*). O parâmetro $thres$ determina o tamanho dos agrupamentos de tarefas durante a formação e análise de lotes candidatos, enquanto κ atua diretamente no cálculo dos índices gerados pelo ATC e BATC-II. Dessa forma, para produzir soluções distintas, o $\text{ILS}_{\text{batch-pm}}$ define esses valores de forma aleatória a cada execução.

4.1.4 Perturbação

Neste estágio do algoritmo são aplicadas operações aleatórias na solução corrente visando contornar possíveis ótimos locais gerados a cada iteração. Essas alterações nos

Algoritmo 3 TWD

input: $thres, \kappa$

1: Quando uma máquina estiver disponível no tempo t, considerar a janela $(t, t + \Delta t)$.
2: Definir o conjunto $M(f, t, \Delta t) = \{j | r_j \leq t + \Delta t, f_j = f\}$ referente às tarefas não alocadas da família f.
3: Ordenar as tarefas do conjunto $M(f, t, \Delta t)$ de forma decrescente em função do índice ATC (4.1).
4: Criar o conjunto $\tilde{M}(f, t, \Delta t, thres) = \{j | j \in M(f, t, \Delta t), rank(j) \leq thres\}$, onde $rank$ se refere à posição da tarefa j no conjunto $M(f, t, \Delta t)$ após a ordenação do passo anterior.
5: Realizar todas as combinações de lotes possíveis com as tarefas presentes no conjunto $\tilde{M}(f, t, \Delta t, thres)$ e analisar cada um segundo o índice BATC-II (4.2).
6: Vincular o lote com maior índice BATC-II à solução.
7: Caso ainda haja tarefas não alocadas, definir uma nova janela de tempo e retornar ao primeiro passo.

componentes da solução devem ser equilibradas, pois mudanças com pouco impacto podem não ser efetivas o suficiente para superar ótimos locais, enquanto alterações maiores podem se comportar como um reinício do processamento, comprometendo a evolução natural dos resultados (Lourenço et al., 2019).

O funcionamento do procedimento de perturbação é apresentado no Algoritmo 4. A quantidade de execuções dos movimentos de perturbação é definida de forma aleatória em função do parâmetro I_{mp}.

Algoritmo 4 Perturbação

input: S, I_{mp}

1: Seja n um número aleatório entre 1 e I_{mp}.
2: **for** $iter = 1$ to n **do**
3: Aplicar operador na solução S.
4: **return** S

Após uma análise detalhada da eficácia de vários operadores de perturbação (ver Tabela 5.2), foi adotado o *Batch Split* (Algoritmo 5). Nessa operação, um lote é selecionado de forma aleatória. Em sequência, seus itens são reorganizados aleatoriamente e divididos em dois novos lotes em um ponto de corte definido de forma randômica. Os dois lotes resultantes são inseridos em posições aleatórias na solução.

Algoritmo 5 *Batch Split*

input: S

1: Obter aleatoriamente um lote L com mais de duas tarefas da solução S.
2: Remover L da solução e reordenar de forma aleatória suas tarefas.
3: Definir o ponto de cisão c de forma aleatória, onde $c \in (1, |L| - 1)$.
4: Criar um novo lote L_1 com as primeiras c tarefas de L.
5: Criar um novo lote L_2 com as demais tarefas de L.
6: Adicionar os novos lotes L_1 e L_2 em posições aleatórias da solução.

4.1.5 Busca local

Nesta fase do algoritmo, denominada intensificação, é empregada a busca local para aprimorar a solução corrente. Neste estágio, assim como no trabalho elaborado por Queiroga et al. (2021), é utilizado o procedimento *Randomized Variable Neighbourhood Descent* (RVND) (Subramanian et al., 2010). Esse método consiste numa variação do *Variable Neighbourhood Descent* (VND), onde a seleção das estruturas de vizinhança para atuar na melhoria da qualidade das soluções ocorre de forma aleatória.

Algoritmo 6 Busca local (RVND)

input: S

1: Seja N o conjunto de todas as estruturas de vizinhança consideradas.
2: **for** $i = 1$ to $|N|$ **do**
3: $\quad S' = S$
4: $\quad$ Aplicar o operador N_i na solução S'.
5: $\quad$ **if** $f(S') < f(S)$ **then**
6: $\quad\quad S = S'$
7: $\quad\quad i = 0$
8: $\quad\quad$ Permutar os elementos de N.
9: **return** S

O Algoritmo 6 demonstra os passos do procedimento da busca local. Inicialmente, é gerada uma lista contendo todas as estruturas de vizinhança ordenadas de forma aleatória (linha 1). Em seguida, cada estrutura é selecionada iterativamente e aplicada na solução corrente na tentativa de aprimorar sua qualidade (linha 4). A implementação de cada um desses mecanismos explora todas as possibilidades de movimento dentro da solução em questão e seleciona aquele que obteve melhor resultado, ou seja, é realizada uma pesquisa exaustiva pela melhor opção. Se houver uma melhoria no

custo associado (linha 5), esse movimento é integrado à solução corrente e uma nova ordem é atribuída à lista de vizinhanças para reinício das iterações. No entanto, não havendo aprimoramento da solução, o procedimento aplicará o próximo componente da lista. Finalmente, quando todas as alternativas se esgotarem sem sucesso, a execução da busca se encerrará.

As estruturas de vizinhança empregadas na busca local adotam como estratégia o critério *best improvement*, onde os movimentos que as caracterizam, e que resultam em uma solução viável, são aplicados sucessivamente na solução corrente, cobrindo, portanto, todas as vizinhanças possíveis. O movimento responsável pela vizinhança que oferecer o menor custo será aplicado à solução.

De forma geral, as estruturas de vizinhança implementadas se baseiam em movimentos relacionados à realocação de lotes e tarefas. Independentemente de qual tipo de movimento ocorra, a solução se ajustará a cada passo dinamicamente a fim de otimizar a operação. Segue demonstrações de cada caso:

- Operações associadas a tarefas: ao se remover uma tarefa da solução, atributos associados ao lote ao qual pertencia, como datas de liberação e entrega, podem ser alterados por estarem vinculados a propriedades das tarefas que o compõem. Havendo alguma mudança nesse sentido, o cronograma de execução da máquina será atualizado a partir do lote em questão. De forma equivalente, ao se incluir uma tarefa em novo lote, poderá ocorrer alteração dos seus atributos e consequente atualização da agenda de processamento da máquina.

- Operações associadas a lotes: quando um lote é removido de sua posição original, a fila posterior a ele, quando possível, será acomodada no espaço deixado em aberto, atualizando, automaticamente, o cronograma de execução da máquina. De forma semelhante, no momento em que um lote é incluído em nova posição, aquele que a ocupava anteriormente e os subsequentes também terão sua agenda de processamento ajustada.

Após estudos com diferentes implementações de estruturas de vizinhança (ver Tabela 5.3), este trabalho adotou os seguintes operadores:

- *Job Insertion*: iterativamente, remove-se cada tarefa do lote ao qual pertence para efetuar sua inserção em todas as posições possíveis da solução. Essa acomodação pode ocorrer tanto em lotes anteriormente existentes como em novos. Neste caso específico, um lote é criado temporariamente para receber a tarefa e, em seguida, é adicionado entre os lotes de cada máquina.

- *Job Swap*: tarefas de uma mesma família pertencentes a diferentes lotes da solução tem suas posições sistematicamente trocadas.

- *Batch Insertion*: cada lote é removido de sua posição original e inserido em todas as outras posições de cada máquina.

- *Batch Swap*: todos os lotes da solução têm suas posições permutadas entre si.

- *Batch Merge*: um lote é sistematicamente agrupado com todos os outros lotes da mesma família, formando, a cada iteração, um novo lote viável. Caso os dois lotes envolvidos na operação estejam na mesma máquina, o novo lote é inserido na posição daquele mais à esquerda no cronograma da máquina, ou seja, aquele programado para ser executado antes. Caso contrário, o novo lote é analisado nas posições originais de cada um dos envolvidos na operação.

As estruturas de vizinhança adotadas possuem tamanho $\mathcal{O}(n^2)$ e são necessárias $\mathcal{O}(n)$ operações para analisar cada movimento, gerando uma complexidade geral de $\mathcal{O}(n^3)$. Uma descrição abrangente dessas vizinhanças, incluindo procedimentos eficientes de avaliação dos movimentos, pode ser encontrada em Queiroga et al. (2021). Apesar de originalmente propostos para a variante com uma máquina, eles podem ser diretamente estendidos para o problema com máquinas paralelas.

4.2 Algoritmo Populacional

O segunda solução desenvolvida neste trabalho, denominada $\text{PILS}_{\text{batch-pm}}$, adiciona uma abordagem populacional ao $\text{ILS}_{\text{batch-pm}}$. O algoritmo incorpora mecanismos de gerenciamento de população, geração e aperfeiçoamento de indivíduos e controle de diversidade. O aprimoramento de soluções se baseia no emprego de mecanismos de

perturbação e procedimento de busca local. Essa nova proposta teve por base a metaheurística *Hybrid Genetic Search with Adaptative Diversity Control* (HGSADC) (Vidal et al., 2012), aplicada com êxito em variantes do Problema de Roteamento de Veículos (PRV).

O Algoritmo 7 apresenta os principais componentes da solução elaborada. São considerados parâmetros de entrada μ, que estabelece o tamanho mínimo da população, λ, que indica o número de descendentes numa geração ou quantidade máxima de indivíduos criados a partir da população mínima, It_{lim}, que corresponde ao total de iterações sem aperfeiçoamento da melhor solução encontrada, μ^{elite}, que especifica a quantidade de itens sobreviventes na diversificação e μ^{close}, que se refere ao número de vizinhos mais próximos considerado no cálculo da contribuição de diversidade de cada indivíduo.

Inicialmente, uma população de tamanho μ é construída com indivíduos gerados de forma aleatória (linha 1). Em seguida, iterativamente, obtém-se um indivíduo da população através do Torneio Binário para ser utilizado como base na produção de um novo (linhas 3 e 4, respectivamente). Posteriormente, são aplicadas perturbações e procedimento de busca local com múltiplas estruturas de vizinhança para permitir seu rápido desenvolvimento (linha 5). Em sequência, o indivíduo aprimorado é adicionado à população (linha 6). Em caso de It_{lim} iterações sem melhora, ocorrerá a diversificação da população com a eliminação dos indivíduos menos aptos e a inclusão de novos (linha 8). Se o tamanho da população estiver em seu valor máximo $\mu + \lambda$, um total de λ indivíduos menos aptos serão descartados da população (linha 10). Finalmente, após o critério de parada ser atendido, o melhor resultado encontrado é retornado.

4.2.1 População inicial

Na construção da população inicial, é adotada a heurística TWD apresentada anteriormente neste trabalho (ver Subseção 4.1.3). Todavia, no âmbito do algoritmo implementado, as variáveis $thres$ e κ, cujos valores são definidos aleatoriamente, são preservados com o indivíduo e transmitidos aos seus descendentes, permitindo seu reaproveitamento em fases futuras do processamento.

Algoritmo 7 Algoritmo $\text{PILS}_{\text{batch-pm}}$

input: μ, λ, It_{lim}, μ^{close}, μ^{elite}

```
1: Gerar uma população inicial P com μ indivíduos
2: while critério de parada não atendido do
3:    Obter indivíduo I pelo Torneio Binário
4:    Criar um novo indivíduo I' igual a I
5:    Perturbar e aplicar busca local em I'
6:    Adicionar I' à população P
7:    if melhor solução não aperfeiçoada em It_lim then
8:       Diversificar população
9:    if |P| = μ + λ then
10:      Selecionar sobreviventes
11: return melhor solução encontrada
```

4.2.2 Formação de novo indivíduo

A formação de um indivíduo se inicia com o Torneio Binário. Esse mecanismo seleciona dois indivíduos da população de forma aleatória e define como vencedor aquele que possuir melhor aptidão. Em seguida, um novo indivíduo é gerado com base no vencedor para, então, ser submetido a procedimentos de perturbação e busca local.

De forma similar ao $\text{ILS}_{\text{batch-pm}}$, a perturbação oferece a possibilidade de contornar ótimos locais gerados a cada iteração. Na implementação adotada no $\text{PILS}_{\text{batch-pm}}$, tanto os passos quanto os parâmetros de entrada são aqueles apresentados no Algoritmo 4. Ainda, pelo mesmo motivo citado anteriormente, foi empregado o operador *Batch Split* (ver Subseção 4.1.4).

Ao término da perturbação, é executado o procedimento de busca local para aprimorar a solução. Para alcançar esse objetivo, foi empregado o mesmo algoritmo desenvolvido para o $\text{ILS}_{\text{batch-pm}}$ (ver Algoritmo 6), assim como a mesma relação de estruturas de vizinhança. Essa configuração se mostrou acertada na resolução do problema, sendo, portanto, reaproveitada.

4.2.3 Gerenciamento da população

O gerenciamento da população destina-se a promover indivíduos com maior aptidão dentro do grupo populacional. Nesse sentido, uma avaliação contínua é realizada para

determinar o nível de adaptação de cada membro ao ambiente. Essa análise relaciona o custo da solução vinculada ao indivíduo e sua contribuição para a diversidade da população.

A contribuição para a diversidade de um indivíduo $\Delta(I)$ (4.3) é definida pela média de sua distância para seus vizinhos mais próximos N_{close}:

$$\Delta(I) = \frac{1}{|N_{close}|} \sum_{I_2 \in N_{close}} \delta(I, I_2), |N_{close}| \leq \mu^{close}. \tag{4.3}$$

O cálculo da distância entre dois indivíduos $\delta(I, I_2)$ se baseia na distância de Hamming e seu processamento pode ser observado no Algoritmo 8. A composição dos lotes entre as duas soluções analisadas é o fator que indica o grau de proximidade entre elas. A distância máxima entre dois indivíduos equivale ao número de tarefas do problema $\delta_{max}(I, I_2) = n$. Esse cenário ocorre quando não há lotes com o mesmo agrupamento de tarefas entre as duas soluções. Por outro lado, a distância mínima $\delta_{min}(I, I_2) = 0$ é obtida quando a composição de todos os lotes das soluções são idênticas, independentemente das posições de suas alocações.

Algoritmo 8 Cálculo da distância entre indivíduos

input: I, I_2
1: $d = n$
2: **for** para cada lote b_i de I **do**
3: **if** se há um lote idêntico a b_i em I_2 **then**
4: $d = d - |b_i|$
5: **return** d

A Tabela 4.1 ilustra o cálculo da distância entre dois indivíduos considerando um problema com apenas uma máquina e um conjunto de tarefas $J = \{j_1, \ldots, j_{10}\}$. Cada linha representa a composição de cada lote gerado e a sequência para processamento. Conforme apresentado, a formação dos lotes dos indivíduos difere somente em relação àqueles que envolvem as tarefas j_3, j_4 e j_{10}. Por essa razão, $\delta(I, I_2) = 3$, ainda que o sequenciamento dos demais lotes seja diferente.

A função de aptidão (4.4) utilizada neste trabalho é semelhante à apresentada por Vidal et al. (2012). O resultado da função indica a qualidade do indivíduo dentro do grupo e seu cálculo ocorre da seguinte forma:

Tabela 4.1: Cálculo da distância entre indivíduos.

Indivíduo	Sequência de lotes					
I	$\{j_1, j_2\}$	$\{\boldsymbol{j_3}, \boldsymbol{j_4}\}$	$\{j_5\}$	$\{j_6, j_7\}$	$\{j_8, j_9\}$	$\{\boldsymbol{j_{10}}\}$
I_2	$\{j_5\}$	$\{\boldsymbol{j_3}, \boldsymbol{j_4}, \boldsymbol{j_{10}}\}$	$\{j_1, j_2\}$	$\{j_8, j_9\}$	$\{j_6, j_7\}$	
$\delta(I, I_2) = 3$						

$$f_{\mathcal{P}}(I) = f_{\mathcal{P}}^{custo}(I) + \left(1 - \frac{\mu^{elite}}{|\mathcal{P}|}\right) f_{\mathcal{P}}^{div}(I). \tag{4.4}$$

O componente $f_{\mathcal{P}}^{custo}(I)$ indica a posição de classificação do indivíduo I em relação aos outros da população com base no custo das soluções, ou seja, define o *rank* do indivíduo quanto ao valor de sua função objetivo. Da mesma forma, a função $f_{\mathcal{P}}^{div}(I)$ é usada para determinar a posição de classificação do indivíduo com base na média das distâncias. No entanto, nesse caso, a classificação é feita em ordem decrescente, favorecendo membros que apresentam uma contribuição mais alta da diversidade. Como resultado, ao realizar alterações na população, ou seja, ao incluir ou eliminar indivíduos, é necessário recalcular as distâncias e a função de aptidão para se ajustarem à nova composição do grupo populacional.

A análise de sobrevivência se baseia no valor da função de aptidão de cada indivíduo e ocorre em dois cenários, quando a população atinge seu limite e quando o número de iterações sem evolução do melhor custo alcançar o valor It_{lim}. No primeiro cenário, quando o grupo populacional totalizar $\mu + \lambda$ indivíduos, aqueles λ menos aptos são eliminados. No segundo caso, após It_{lim} iterações sem melhora, aplica-se o procedimento de diversificação. Nesse momento, todos os indivíduos que não compreendem N_{elite} são descartados, ou seja, apenas os μ^{elite} mais aptos permanecem. A recomposição da população ocorre gerando $\mu - \mu^{elite}$ novos indivíduos com uso do TWD. Neste estágio, antes da criação de cada novo indivíduo, há a escolha aleatória de um dos sobreviventes. Em seguida, a informação dos valores dos parâmetros $thres$ e κ, disseminados até ele através dos seus ascendentes, são resgatados. Os novos parâmetros $thres'$ e κ' assumem os possíveis valores: $thres' = thres - 1, \ldots, thres + 1$ e $\kappa' = \kappa - 0.1, \ldots, \kappa + 0.1$. Dessa forma, são introduzidos parâmetros próximos daqueles responsáveis por

gerar uma descendência de alta qualidade.

Capítulo 5

Resultados computacionais

Este capítulo tem por objetivo apresentar o ambiente de processamento, a calibração dos parâmetros e os resultados dos experimentos dos algoritmos propostos em comparação com outros obtidos na literatura.

Os experimentos foram conduzidos em máquinas do tipo *c6a.large* da *Amazon Web Services* (AWS), configuradas com processador AMD EPYC 7R13 3.6 GHz e com 4 GB de memória RAM e ambientadas com o sistema operacional Ubuntu 22.04. A solução especificada neste trabalho foi implementada na linguagem de programação C++ (versão 17).

As instâncias consideradas foram aquelas produzidas por Chiang et al. (2010), que estabeleceu 486 configurações para o problema combinando os parâmetros apresentados na Tabela 5.1. Para cada uma dessas combinações de parâmetros foram geradas 10 instâncias, totalizando 4860 variações. Para um maior aprofundamento, Mönch et al. (2005) fornece maiores detalhes sobre os parâmetros destacados.

O critério de parada adotado nas duas propostas apresenta a mesma condição imposta por Fidelis e Arroyo (2017), que considerou um limite de tempo igual a $t \times n$ segundos, onde t se refere a uma constante e n o número de tarefas da instância. Neste trabalho, foi atribuído o valor 0,1 para t no processamento de todos os experimentos.

A calibração dos parâmetros e a definição dos operadores de vizinhança empregados nos métodos propostos foram realizadas por meio de testes com instâncias do problema

Tabela 5.1: Parâmetros das instâncias.

Parâmetros	**Valores**
Número de famílias (f)	3, 6, 12
Número de máquinas (m)	3, 4, 5
Número de tarefas (n)	180, 240, 300
Capacidade máxima do lote (Q)	4, 8
Tempo de processamento da família (p_j)	2, 4, 10, 16, 20
Prioridade da tarefa (w_j)	Uniforme (0, 1)
Data de liberação da tarefa (r_j)	$r_j \sim$ Uniforme $(0, \alpha/(mQ)\sum p_j)$ $\alpha = 0.25, 0.5, 0.75$
Data de entrega da tarefa (d_j)	$d_j - r_j \sim$ Uniforme $(0, \beta/(mQ)\sum p_j)$ $\beta = 0.25, 0.5, 0.75$

e os resultados avaliados em termos de qualidade e desempenho. As seções posteriores apresentam os procedimentos executados e os resultados alcançados na condução dessa análise. Por último, é apresentado o desempenho dessas soluções em comparação com algoritmos de outros autores.

5.1 Calibração do TWD

Em relação aos parâmetros do TWD, heurística responsável por construir soluções iniciais nos dois métodos propostos, foram adotados valores próximos àqueles praticados por Bilyk et al. (2014), que empregaram $thres = 15$ e $\kappa = 0.1 \times h$, $h = 1, \ldots, 50$. Nesse estudo, o TWD é executado iterativamente, variando κ dentro do intervalo especificado, e o melhor resultado aproveitado. Essa estratégia foi adotada por Fidelis e Arroyo (2017) com mesma finalidade. Nesse contexto, após testes em um grupo de instâncias reduzido, este trabalho adotou $thres = 10, \ldots, 20$ para manter a proximidade do nível de qualidade e de desempenho. Constatou-se que o custo das soluções foi afetado negativamente quando $thres < 10$, enquanto o esforço computacional se elevou consideravelmente para $thres > 15$, possivelmente, em razão do aumento de combinações na formação de lotes candidatos. Em relação ao parâmetro κ, foram analisados os dados das execuções do TWD em todas as instâncias variando seu valor no intervalo praticado por Bilyk et al. (2014) e fixando $thres = 15$. Como resultado, adotou-se

$\kappa = 0.1 \times h$, $h = 5, \ldots, 25$, pois 89,83% dos melhores custos se concentraram nessa faixa de valores.

5.2 Seleção dos operadores de vizinhança

Diferentes estruturas de vizinhança e mecanismos de perturbação foram estudados e aqueles que apresentaram melhor eficácia foram selecionados. Além dos operadores presentes nas soluções baseadas no VNS, IG e SA, também foram consideradas as estruturas desenvolvidas por Queiroga et al. (2021), que elaboraram um algoritmo eficiente apresentando ótimos resultados para o problema de escalonamento de lotes em uma máquina.

Na avaliação desses procedimentos, foram realizados testes em uma versão simplificada do $\text{ILS}_{\text{batch-pm}}$, desconsiderando reinícios em função da quantidade de iterações sem melhora. O algoritmo foi configurado para selecionar aleatoriamente um único operador a cada iteração. Os experimentos foram realizados em 486 instâncias do problema escolhidas de forma aleatória, sendo uma de cada cenário possível.

Inicialmente, foram conduzidos testes para avaliar os operadores de perturbação. Nessa análise, como ainda não haviam sido definidos os procedimentos de busca local, foram adotados aqueles sugeridos por Queiroga et al. (2021), com as devidas adaptações para o problema com máquinas em paralelo.

Como a perturbação é executada antes da busca local para facilitar a superação de ótimos locais, foi estabelecido como objeto da análise a capacidade de cada operador em cumprir esse objetivo. A Tabela 5.2 apresenta os resultados dos testes dos diferentes mecanismos implementados. Quando a execução do algoritmo é iniciada, $i = 0$. Se, ao final da primeira iteração, a melhor solução não for aprimorada, então i será incrementado. Posteriormente, caso a solução seja melhorada, i será zerado. Em resumo, o contador i indica a quantidade de iterações processadas desde a última atualização da melhor solução. A segunda coluna ($i \geq 0$) contabiliza todos os aprimoramentos de um operador e os relaciona com todas as suas execuções, trata-se, portanto, da sua eficácia geral. As demais colunas restringem os valores de i. Assim, são consideradas somente as execuções do operador quando i estiver dentro do intervalo indicado. Por exemplo,

na terceira coluna, o movimento de *Batch Split* assume o valor 15,19, pois obteve 1081 melhorias quando executado em 7118 ocasiões com i variando de 0 a 10. Com base nos dados compilados, foi selecionado o mecanismo *Batch Split* por apresentar a maior eficácia entre os operadores listados.

Em seguida, foram conduzidos testes adicionais para determinar o número máximo de movimentos de perturbação a serem executados em cada iteração. Foram considerados limites variando de 1 até 4. Como resultado, foi definido $I_{mp} = 3$, pois esse valor se mostrou o mais favorável no intervalo investigado. Portanto, a cada iteração, até 3 movimentos de perturbação serão realizados, sendo a quantidade determinada de forma aleatória.

No estudo dos procedimentos da busca local, foram executados testes semelhantes aos realizados na análise dos operadores de perturbação. Neste caso, o procedimento de perturbação observou os passos descritos no Algoritmo 4 e adotou o operador selecionado anteriormente. A Tabela 5.3 apresenta a eficácia de cada estrutura de vizinhança em melhorar a qualidade de uma solução submetida ao seu processamento. Com base nos dados coletados, foram selecionadas aquelas com resultados superiores, descartando estruturas com mesma base de funcionamento. Assim sendo, foram incorporados os mecanismos *Batch Insertion*, *Job Swap*, *Batch Swap*, *Job Insertion* e *Batch Merge*.

5.3 Calibração do $ILS_{batch\text{-}pm}$ e do $PILS_{batch\text{-}pm}$

No contexto do $ILS_{batch\text{-}pm}$, após a definição dos operadores de vizinhança e do limite de movimentos de perturbação (I_{mp}), restou determinar o valor do parâmetro I_{ils}, que controla o número máximo de iterações sem melhora. Uma análise dos dados apresentados na Tabela 5.2 revela uma queda acentuada na eficácia do algoritmo à medida que o contador i é incrementado sucessivamente. Com base nessa observação, decidiu-se estabelecer $I_{ils} = 100$, pois valores superiores indicam maior dificuldade em superar ótimos locais, com consequente desperdício de recurso computacional.

Acerca do $PILS_{batch\text{-}pm}$, foram adotados o mesmo mecanismo de perturbação e as mesmas estruturas de vizinhança incorporados ao $ILS_{batch\text{-}pm}$. Os valores dos parâmetros relacionados com a população, μ e λ, foram determinados após testes preliminares

Tabela 5.2: Resultados percentuais da eficácia dos operadores de perturbação.

Operador	Fonte	Total de melhorias com i no intervalo / total de execuções com i no intervalo						
		$i \geq 0$	$i \in [0,10]$	$i \in [11,50]$	$i \in [51,100]$	$i \in [101,250]$	$i \in [251,500]$	$i \geq 500$
Batch Split	Fidelis e Arroyo (2017)	**5,01**	**15,19**	**4,22**	**2,03**	**0,73**	0,17	0,03
Job Swap	Queiroga et al. (2021)	4,04	14,96	1,59	0,54	0,22	0,17	0,10
Batch Merge	Fidelis e Arroyo (2017)	4,39	14,67	2,80	1,12	0,54	0,15	0,10
Batch Split	Queiroga et al. (2021)	4,34	14,51	2,77	1,01	0,51	0,12	0,06
Batch Swap	Queiroga et al. (2021)	4,51	14,12	3,14	1,73	0,68	**0,38**	0,03
Job Insertion	Queiroga et al. (2021)	4,17	14,10	2,43	1,16	0,41	0,03	0,00
Batch Swap	Fidelis e Arroyo (2017)	4,39	14,07	3,16	1,33	0,52	0,17	**0,16**
Move Batch ($l = 1$)	Bilyk et al. (2014)	4,54	13,64	3,81	1,79	0,58	0,18	0,09
Batch Insertion	Queiroga et al. (2021)	4,21	13,12	3,12	1,41	0,58	0,33	0,10

Tabela 5.3: Resultados da eficácia dos operadores da busca local.

Operador	Fonte	Execuções	Melhorias	Eficácia (%)
Batch Insertion	Queiroga et al. (2021)	501039	412166	82,26
Job Swap	Queiroga et al. (2021), Bilyk et al. (2014)	522332	427384	81,82
Batch Swap	Queiroga et al. (2021)	490822	369114	75,20
Batch Swap	Bilyk et al. (2014)	498413	366121	73,46
Job Insertion		508092	365517	71,94
Job Insertion	Bilyk et al. (2014)	527590	355836	67,45
Job Insertion	Queiroga et al. (2021)	511501	318511	62,27
Batch Merge	Queiroga et al. (2021)	484973	247036	50,94
Batch Insertion	Fidelis e Arroyo (2017)	499281	105977	21,23

em um subconjunto de instâncias. Resultados promissores foram observados com $\mu = 5$ e $\lambda = 10$. Os demais parâmetros populacionais foram fixados de acordo com Vidal et al. (2012), assim, $\mu_{elite} = 0.4 \times \mu = 2$ e $\mu_{close} = 0.2 \times \mu = 1$. Ainda, definiu-se $It_{lim} = 250$ após experimentos com diferentes valores, quais sejam, 250, 500 e 1000.

Finalmente, a definição do modo de criação de um novo indivíduo durante o estágio de diversificação também foi objeto de análise. Foram considerados o método sugerido por Vidal et al. (2012), criando um novo indivíduo de forma aleatória, como acontece na construção da população inicial, e o mecanismo de reaproveitamento dos parâmetros $thres$ e κ, descrito na Subseção 4.2.3. Nessa avaliação, os dois métodos foram submetidos a testes em todas as instâncias do problema. A segunda abordagem obteve melhor resultado, alcançando 55% dos melhores resultados e 53% das melhores médias. Esse comportamento pode ser justificado pela necessidade de rápida convergência em razão do pouco tempo destinado ao processamento do algoritmo. Assim, novos indivíduos gerados a partir de parâmetros prósperos tendem a evoluir mais rapidamente.

A Tabela 5.4 elenca todos os parâmetros empregados nos dois métodos propostos neste trabalho.

5.4 Comparação com algoritmos existentes

Para viabilizar esta análise, foram implementados os algoritmos com melhores resultados reportados na literatura para o problema $P_m|r_j, batch, incompatible|\sum w_j T_j$,

Tabela 5.4: Parâmetros do $\text{ILS}_{\text{batch-pm}}$ e $\text{PILS}_{\text{batch-pm}}$.

Parâmetros		Valores
$thres$	Tamanho agrupamento de tarefas	$[10, 20]$
κ	*Look-ahead* $(0.1 \times h, h \in [5, 25])$	$[0.5, 2.5]$
I_{ils}	Máximo de iterações sem melhora	100
I_{mp}	Máximo de perturbações por iteração	3
μ	Tamanho mínimo da população	5
λ	Número de descendentes numa geração	10
It_{lim}	Máximo de iterações sem melhora na abordagem populacional	250
μ_{elite}	Número de indivíduos elite $(0.4 \times \mu)$	2
μ_{close}	Vizinhos mais próximos $(0.2 \times \mu)$	1

quais sejam, o método baseado no VNS, proposto por Bilyk et al. (2014), e aqueles com base no IG e no SA, relatados por Fidelis e Arroyo (2017). A construção dessas soluções obedeceu criteriosamente todos os passos, regras e parâmetros apresentados, incluindo aqueles presentes em trabalhos por eles referenciados. Importante destacar que toda a estrutura de dados, implementações de vizinhanças, quando idênticas, mecanismos de cálculo de resultados e demais componentes computacionais desenvolvidos para o $\text{ILS}_{\text{batch-pm}}$ e $\text{PILS}_{\text{batch-pm}}$ foram empregados na construção dessas soluções. Portanto, a aferição dos resultados se dá, efetivamente, pelas decisões estratégicas de cada proposta.

Todas as instâncias foram executadas dez vezes por cada método em análise e os resultados compilados e comparados com uso da métrica *Relative Percentage Improvement* (RPI), que quantifica de forma relativa os resultados alcançados por uma solução em relação àqueles produzidos por outra. Neste trabalho, adota-se como referência os valores obtidos com uso do TWD conforme o método sugerido por Bilyk et al. (2014). As fórmulas a seguir permitem que os RPIs sejam calculados em grupos de instâncias:

$$RPI^b(alg) = 1 - \frac{1}{|I|} \sum_{u=1}^{|I|} \left(\frac{\min\limits_{k=1,\ldots,10} \{TWT_u(alg, k)\}}{TWT_u(twd)} \right) \tag{5.1}$$

$$RPI^m(alg) = 1 - \frac{1}{|I|} \sum_{u=1}^{|I|} \left(\frac{\sum_{k=1}^{10} (TWT_u(alg, k)/10)}{TWT_u(twd)} \right) \tag{5.2}$$

$$RPI^{w}(alg) = 1 - \frac{1}{|I|}\sum_{u=1}^{|I|}\left(\frac{\max\limits_{k=1,\ldots,10}\{TWT_u(alg,k)\}}{TWT_u(twd)}\right) \tag{5.3}$$

As equações (5.1), (5.2) e (5.3) consideram em seus cálculos o melhor valor encontrado, a média dos resultados e o pior valor para cada instância, respectivamente. O termo *alg* denota o algoritmo analisado. O parâmetro I representa o grupo de instâncias considerado e u o índice dos itens que o compõe. A variável k se refere às execuções realizadas. Portanto, $TWT_u(alg, k)$ representa o custo associado com aplicação do algoritmo alg na instância u na rodada k. Os grupos examinados, instâncias que compartilham um determinado parâmetro de sua configuração em comum, e a compilação dos resultados estão sintetizados na Tabela 5.5.

Como apresentado, os desempenhos do $\text{ILS}_{\text{batch-pm}}$ e do $\text{PILS}_{\text{batch-pm}}$ superaram os demais, considerados os melhores da literatura, em todos os cenários delineados de forma sistemática. Os resultados foram robustos e consistentes, permitindo obter os melhores valores em todas as métricas consideradas. Ainda, o $\text{ILS}_{\text{batch-pm}}$ demonstrou atuação melhor em relação ao $\text{PILS}_{\text{batch-pm}}$, no entanto, por uma margem pequena.

A Tabela 5.6 aponta a quantidade de melhores resultados e melhores médias que cada método conseguiu alcançar em cada grupo de instâncias do problema. Novamente, os algoritmos propostos demonstraram facilidade em superar as demais soluções. No entanto, da mesma forma como evidenciado na Tabela 5.5, os dois métodos mostraram resultados próximos. Em relação à média de todos os resultados processados, ou seja, das 4860×10 execuções de cada proposta, o $\text{ILS}_{\text{batch-pm}}$ obteve o valor 864,58, enquanto o $\text{PILS}_{\text{batch-pm}}$ obteve 865,04. Possuem, portanto, desempenho equilibrado, sem que um deles tenha domínio sobre o outro.

Tabela 5.5: Resultados percentuais com base nas equações (5.1), (5.2) e (5.3).

Grupo	VNS			IG			SA			$ILS_{batch\text{-}pm}$			$PILS_{batch\text{-}pm}$		
Famílias															
3	36.16	34.02	31.55	25.48	22.42	19.21	19.14	15.76	12.39	**38.45**	**36.97**	**35.10**	38.14	36.80	35.08
6	32.73	30.69	28.45	22.11	19.16	16.02	18.27	14.90	11.57	**35.09**	33.70	32.00	35.04	**33.73**	**32.06**
12	30.54	28.58	26.44	20.48	17.62	14.61	19.37	16.17	12.98	32.72	31.42	**29.88**	**32.78**	**31.49**	29.82
Máquinas															
3	36.18	33.79	31.15	25.45	22.17	18.70	22.37	18.67	14.97	**39.09**	**37.50**	35.54	38.95	37.45	**35.57**
4	32.91	30.89	28.64	22.43	19.52	16.44	18.50	15.23	11.96	**35.07**	**33.72**	**32.05**	34.98	33.68	32.01
5	30.34	28.62	26.65	20.19	17.51	14.70	15.91	12.92	10.01	**32.11**	30.87	**29.39**	32.04	**30.88**	29.38
Capacidade lote															
4	42.99	41.02	38.83	29.98	26.51	22.75	26.13	22.07	17.98	**45.46**	**44.05**	**42.36**	45.29	43.97	42.33
8	23.30	21.18	18.80	15.40	12.96	10.47	11.73	9.15	6.65	**25.38**	24.01	22.29	25.35	**24.04**	**22.31**
Número tarefas															
180	28.36	26.66	24.65	20.99	18.22	15.38	17.86	14.70	11.57	29.68	28.75	**27.58**	**29.73**	**28.76**	27.41
240	33.43	31.27	28.91	22.75	19.77	16.65	18.70	15.36	12.01	**35.82**	34.37	32.62	35.75	**34.39**	**32.65**
300	37.64	35.37	32.87	24.33	21.22	17.81	20.22	16.76	13.37	**40.77**	**38.97**	36.77	40.48	38.86	**36.90**
Datas de liberação															
0.25	29.09	27.68	26.03	19.15	16.89	14.57	19.11	16.65	14.14	**30.36**	**29.46**	**28.30**	30.29	29.43	28.25
0.5	34.46	32.34	29.96	23.33	20.23	16.98	19.59	16.18	12.80	**36.83**	**35.35**	33.52	36.72	35.31	**33.53**
0.75	35.88	33.28	30.45	25.59	22.08	18.29	18.07	13.99	10.01	**39.07**	**37.29**	35.15	38.95	37.28	**35.19**
Datas de entrega															
0.25	20.52	18.92	17.14	13.06	11.04	8.92	10.39	8.19	6.07	**22.24**	21.11	19.69	22.17	**21.12**	**19.78**
0.5	34.91	32.80	30.47	23.78	20.77	17.66	20.49	17.19	13.90	**37.30**	**35.84**	**34.08**	37.13	35.77	34.04
0.75	44.00	41.58	38.83	31.23	27.39	23.25	25.90	21.44	16.98	**46.73**	**45.14**	**43.20**	46.66	45.12	43.14
Média geral	33.14	31.10	28.81	22.69	19.73	16.61	18.93	15.61	12.32	**35.42**	**34.03**	**32.32**	35.32	34.01	**32.32**

Tabela 5.6: Totais de melhores resultados e médias por cenário.

Grupo	VNS	IG	SA	$ILS_{batch\text{-}pm}$	$PILS_{batch\text{-}pm}$
Famílias					
3	97 27	1 0	0 0	**1066 948**	681 653
6	37 2	0 0	0 0	**884** 757	878 **864**
12	31 3	0 0	0 0	769 763	**992 859**
Máquinas					
3	36 13	1 0	0 0	**954 862**	804 753
4	49 8	0 0	0 0	**896 821**	866 796
5	80 11	0 0	0 0	869 785	**881 827**
Capacidade lote					
4	37 11	1 0	0 0	**1404 1317**	1081 1107
8	128 21	0 0	0 0	1315 1151	**1470 1269**
Número tarefas					
180	120 20	0 0	0 0	859 802	**1110 809**
240	24 6	1 0	0 0	**876** 781	800 **835**
300	21 6	0 0	0 0	**984 885**	641 732
Datas de liberação					
0.25	103 25	0 0	0 0	**914 836**	878 768
0.5	45 5	0 0	0 0	**910 839**	817 778
0.75	17 2	1 0	0 0	**895** 793	856 **830**
Datas de entrega					
0.25	72 17	0 0	0 0	**926** 788	860 **822**
0.5	49 8	0 0	0 0	**897 833**	840 782
0.75	44 7	1 0	0 0	**896 847**	851 772
Total geral	165 32	1 0	0 0	**2719 2468**	2551 2376

Capítulo 6

Considerações finais

Este trabalho abordou uma das variantes do problema de escalonamento de lotes, sendo consideradas tarefas com penalidades por atraso, datas de liberação e de entrega e famílias incompatíveis em um ambiente com máquinas paralelas idênticas visando minimizar a soma dos atrasos ponderados. Para resolver o problema em questão, foram propostos dois métodos baseados na meta-heurística ILS, um deles empregando uma estratégia populacional. Em ambos os casos, foram utilizados operadores de vizinhança eficientes nos estágios de perturbação e busca local. Em relação ao algoritmo populacional, foram introduzidos mecanismos inteligentes para geração de novos indivíduos e de controle de diversidade.

No escopo deste trabalho foram conduzidos experimentos para determinar a escolha dos valores dos parâmetros das soluções propostas, assim como definir os operadores de vizinhança. Ainda, foram implementados os algoritmos que apresentaram os melhores resultados relatados na literatura. Essas implementações foram motivadas pela ausência de informações detalhadas dos resultados publicados. Assim, foi possível realizar uma análise comparativa com objetivo de avaliar o desempenho das diferentes abordagens em um mesmo ambiente computacional.

Os desempenhos apresentados pelos algoritmos desenvolvidos, $\text{ILS}_{\text{batch-pm}}$ e $\text{PILS}_{\text{batch-pm}}$, foram sólidos e substanciais, superando os demais métodos nos diversos grupos de instâncias analisados. Esse resultado evidencia a eficácia das estruturas

de vizinhança adotadas, permitindo alcançar resultados satisfatórios mesmo utilizando abordagens de resolução distintas.

O tempo reduzido de processamento das instâncias não permitiu que todo o potencial do algoritmo $\text{PILS}_{\text{batch-pm}}$ fosse explorado. Como resultado dessa restrição, foram adotados grupos populacionais menores, reduzindo a amplitude da pesquisa e a capacidade de desenvolver de forma mais adequada os indivíduos gerados. Apesar dessas limitações, o $\text{PILS}_{\text{batch-pm}}$ obteve resultados promissores.

Ainda em função desse aspecto, heurísticas empregadas na geração de soluções iniciais precisam entregar bons resultados em pouco tempo de processamento. Caso contrário, haverá um maior desafio dos métodos em aprimorar soluções de menor qualidade em um tempo ainda mais escasso. Diante desse cenário, o uso do TWD se mostrou acertado.

A abundância de instâncias do problema geradas por Chiang et al. (2010) é um fator que limita experimentos mais demorados. Dessa forma, sugere-se que trabalhos futuros revisem essa quantidade e adotem novos critérios de parada para avaliação dos métodos.

Finalmente, em decorrência dos resultados alcançados, trabalhos futuros podem ampliar o emprego dos métodos propostos para as demais classes do problema. As abordagens relatadas são versáteis e podem ser aplicadas com eficácia em diversos cenários.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEDER, C.; MÖNCH, L. Metaheuristics for scheduling jobs with incompatible families on parallel batching machines. *Journal of the Operational Research Society*, v. 62, p. 2083–2096. ISSN 0160-5682, 2011.

BALASUBRAMANIAN, H.; MÖNCH, L.; FOWLER, J.; PFUND, M. Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. *International Journal of Production Research*, v. 42, p. 1621–1638. ISSN 0020-7543, 2004.

BILYK, A.; MÖNCH, L.; ALMEDER, C. Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. *Computers & Industrial Engineering*, v. 78, p. 175–185, 2014.

BLAZEWICZ, J.; ECKER, K. H.; PESCH, E.; SCHMIDT, G.; STERNA, M.; WEGLARZ, J. *Handbook on Scheduling*. Springer International Publishing, Cham. ISBN 978-3-319-99848-0, 2019.

CHIANG, T.-C.; CHENG, H.-C.; FU, L.-C. A memetic algorithm for minimizing total weighted tardiness on parallel batch machines with incompatible job families and dynamic job arrival. *Computers & Operations Research*, v. 37, n. 12, p. 2257–2269. ISSN 03050548, 2010.

FANG, J.; CHEANG, B.; LIM, A. Problems and Solution Methods of Machine Scheduling in Semiconductor Manufacturing Operations: A Survey. *Sustainability*, v. 15, n. 17, p. 13012. ISSN 2071-1050. doi: 10.3390/su151713012, 2023.

FIDELIS, M. B.; ARROYO, J. E. C. Meta-heuristic algorithms for scheduling on parallel batch machines with unequal job ready times. *2017 IEEE International*

Conference on Systems, Man, and Cybernetics (SMC), p. 542–547. IEEE. ISBN 978-1-5386-1645-1, 2017.

FOWLER, J. W.; MÖNCH, L. A survey of scheduling with parallel batch (p-batch) processing. *European Journal of Operational Research*, v. 298, p. 1–24. ISSN 03772217, 2022.

GRAHAM, R. L.; LAWLER, E. L.; LENSTRA, J. K.; KAN, A. R. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, volume 5, p. 287–326. Elsevier, 1979.

KLEMMT, A.; WEIGERT, G.; ALMEDER, C.; MONCH, L. A comparison of mip-based decomposition techniques and vns approaches for batch scheduling problems. *Proceedings of the 2009 Winter Simulation Conference (WSC)*, p. 1686–1694. IEEE. ISBN 978-1-4244-5770-0, 2009.

KRAMER, A.; SUBRAMANIAN, A. A unified heuristic and an annotated bibliography for a large class of earliness–tardiness scheduling problems. *Journal of Scheduling*, v. 22, p. 21–57. ISSN 1099-1425, 2019.

LAUSCH, S.; MÖNCH, L. Metaheuristic approaches for scheduling jobs on parallel batch processing machines. *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, p. 187–207. Springer, 2016.

LAWLER, E. L. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of discrete Mathematics*, volume 1, p. 331–342. Elsevier, 1977.

LEE, C.-Y.; UZSOY, R.; MARTIN-VEGA, L. A. Efficient Algorithms for Scheduling Semiconductor Burn-In Operations. *Operations Research*, v. 40, n. 4, p. 764–775. ISSN 0030-364X, 1992.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search: Framework and applications. *Handbook of metaheuristics*, p. 129–168. Springer, 2019.

MATHIRAJAN, M.; BHARGAV, V.; RAMACHANDRAN, V. Minimizing total weighted tardiness on a batch-processing machine with non-agreeable release times and

due dates. *International Journal of Advanced Manufacturing Technology*, v. 48, n. 9, p. 1133–1148. ISSN 02683768, 2010.

MÖNCH, L.; BALASUBRAMANIAN, H.; FOWLER, J. W.; PFUND, M. E. Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers & Operations Research*, v. 32, n. 11, p. 2731–2750. ISSN 0305-0548, 2005.

MÖNCH, L.; ZIMMERMANN, J.; OTTO, P. Machine learning techniques for scheduling jobs with incompatible families and unequal ready times on parallel batch machines. *Engineering Applications of Artificial Intelligence*, v. 19, n. 3, p. 235–245. ISSN 0952-1976, 2006.

OZTURK, O.; ESPINOUSE, M.-L.; MASCOLO, M. D.; GOUIN, A. Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates. *International Journal of Production Research*, v. 50, n. 20, p. 6022–6035. doi: 10.1080/00207543.2011.641358, 2012.

PINEDO, M. *Scheduling: theory, algorithms, and systems*. Springer, 5 edição. ISBN 978-3-319-26580-3, 2018.

QUEIROGA, E.; PINHEIRO, R. G. S.; CHRIST, Q.; SUBRAMANIAN, A.; PESSOA, A. A. Iterated local search for single machine total weighted tardiness batch scheduling. *Journal of Heuristics*, v. 27, p. 353–438. ISSN 1572-9397, 2021.

SUBRAMANIAN, A.; DRUMMOND, L. M. D. A.; BENTES, C.; OCHI, L. S.; FARIAS, R. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, v. 37, n. 11, p. 1899–1911, 2010.

VAN DER ZEE, D. J.; VAN HARTEN, A.; SCHUUR, P. Dynamic job assignment heuristics for multi-server batch operations- A cost based approach. *International Journal of Production Research*, v. 35, n. 11, p. 3063–3094. ISSN 0020-7543. doi: 10.1080/002075497194291, 1997.

VENKATARAMANA, M.; RAGHAVAN, N. S. Ant colony-based algorithms for scheduling parallel batch processors with incompatible job families. *International Journal of Mathematics in Operational Research*, v. 2, n. 1, p. 73–98, 2010.

VEPSALAINEN, A. P. J.; MORTON, T. E. Priority rules for job shops with weighted tardiness costs. *Management Science*, v. 33, p. 1035–1047. ISSN 0025-1909, 1987.

VIDAL, T.; CRAINIC, T. G.; GENDREAU, M.; LAHRICHI, N.; REI, W. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, v. 60, p. 611–624. ISSN 0030-364X, 2012.

VIMALA RANI, M.; MATHIRAJAN, M. A state-of-art review and a simple meta-analysis on deterministic scheduling of diffusion furnaces in semiconductor manufacturing. *International Journal of Production Research*, v. 0, n. 0, p. 1–28, 2022.

ZHANG, H.; LI, K.; HONG JIA, Z.; CHU, C. Minimizing total completion time on non-identical parallel batch machines with arbitrary release times using ant colony optimization. *European Journal of Operational Research*, v. 309, n. 3, p. 1024–1046. ISSN 0377-2217, 2023.