RICARDO GONÇALVES TAVARES

META-HEURÍSTICAS PARA O SEQUENCIAMENTO DE FAMÍLIAS DE TAREFAS EM MÁQUINAS PARALELAS UNIFORMES DE PROCESSAMENTO EM LOTE

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

Orientador: José Elias Claudio Arroyo

VIÇOSA - MINAS GERAIS
2020

Ficha catalográfica elaborada pela Biblioteca Central da Universidade Federal de Viçosa - Campus Viçosa

T
T231m
2020

Tavares, Ricardo Gonçalves, 1988-
Meta-heurísticas para o sequenciamento de famílias de tarefas em máquinas paralelas uniformes de processamento em lote / Ricardo Gonçalves Tavares. – Viçosa, MG, 2020.
61 f. : il. (algumas color.) ; 29 cm.

Orientador: José Elias Claudio Arroyo.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 58-61.

1. Programação heurística. 2. Processamento eletrônico de dados - Processamento em lote. 3. Otimização combinatória. 4. Algoritmos paralelos. 5. Programação paralela (Computação). 6. Processamento paralelo (Computadores). I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. II. Título.

CDD 22. ed. 006.3

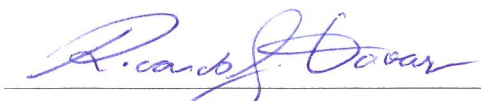
RICARDO GONÇALVES TAVARES

META-HEURÍSTICAS PARA O SEQUENCIAMENTO DE FAMÍLIAS DE TAREFAS EM MÁQUINAS PARALELAS UNIFORMES DE PROCESSAMENTO EM LOTE

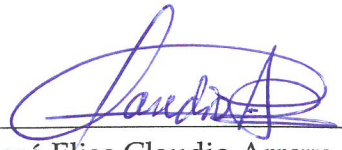
Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 6 de fevereiro de 2020.

Assentimento:



Ricardo Gonçalves Tavares
Autor



José Elias Claudio Arroyo
Orientador

Dedico este trabalho a Deus PRIMEIRAMENTE e aos MEUS pais Nedy e Vera.

Agradecimentos

Agradeço primeiramente a Deus por ser luz, fortaleza e proteção em muitos momentos em minha vida. E pela força para concluir mais essa etapa.

Agradeço aos meus pais Nedy e Vera, pela educação, por todo apoio e incentivo nos meus estudos, pelo amor incondicional e pelo exemplo de generosidade, honestidade e responsabilidade.

Agradeço aos meus irmãos Flávia e Carlos, por estarem sempre torcendo e orando por mim, assim como me ajudando em muitos momentos.

Agradeço também ao meu orientador, Professor José Elias Arroyo, pela paciência e ensina- mentos, e aos demais professores e funcionários do Departamento de Informática da UFV, que contribuíram para minha formação e crescimento profissional.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (Capes) – Código de Financiamento 001.

Agradeço também à Fundação de Amparo à Pesquisa do Estado de Minas Gerais (Fapemig), pelo financiamento da minha bolsa durante o mestrado.

Tente UMA, duas, três vezes e, se possível, tente a quarta, a quinta e quantas vezes for necessário. Só não desista nas PRIMEIRAS tentativas, a persistência é AMIGA da conquista. Se você quer chegar aonde a MAIORIA não chega, faça o que a MAIORIA não faz.

Bill Gates

Resumo

TAVARES, Ricardo Gonçalves, M.Sc., Universidade Federal de Viçosa, fevereiro de 2020. **Meta-heurísticas para o sequenciamento de famílias de tarefas em máquinas paralelas uniformes de processamento em lote**. Orientador: José Elias Claudio Arroyo.

Este trabalho aborda o problema de sequenciar um conjunto de n tarefas com incompatibilidade de famílias, com tamanhos arbitrários e tempos de processamento distintos, em um conjunto de m máquinas paralelas uniformes com capacidades diferentes. Neste problema, as máquinas têm uma característica especial, processar um lote de tarefas simultaneamente, desde que a capacidade da máquina não seja excedida. Apenas tarefas de mesma família podem ser agrupadas em um lote. O tempo de processamento do lote é igual ao maior tempo de processamento entre todas as tarefas do lote. O problema consiste em agrupar as tarefas em lotes e, em seguida, sequenciar os lotes nas máquinas de tal maneira que o tempo de conclusão de todos os lotes seja minimizado (minimização do *makespan*). Como o problema pertence à classe NP-Difícil, neste trabalho, são propostos: i) um modelo de Programação Inteira-Mista para obter soluções ótimas para instâncias pequenas do problema; e ii) dois algoritmos heurísticos baseados em meta-heurísticas para obter soluções de alta qualidade e em tempo computacional aceitável para instâncias de problemas de grande porte. Os algoritmos são baseados nas meta-heurísticas *Iterated Greedy* (IG) e *Discrete Differential Evolution* (DDE). Também estas meta-heurísticas são hibridizadas fazendo uma combinação com métodos de busca local, utilizando uma seleção aleatória de vizinhanças. Os resultados dos experimentos mostram que os desempenhos dos algoritmos propostos são de alta qualidade, tendo o algoritmo DDE-Híbrido apresentado os melhores resultados em comparação aos algoritmos da literatura.

Palavras-chave: Sequenciamento de tarefas. Máquinas paralelas de processamento em lote. Incompatibilidade de família de tarefas. Otimização combinatória. Heurísticas. Meta-heurísticas.

Abstract

TAVARES, Ricardo Gonçalves, M.Sc., Universidade Federal de Viçosa, February, 2020. **Meta-heuristics for the scheduling of job families on uniform parallel batch processing machines**. Advisor: José Elias Claudio Arroyo.

This work addresses the problem of scheduling a set of family incompatible jobs, with arbitrary sizes and different processing times, into a set of uniform parallel machines with different capacities. In this problem the machines have a special feature that is to process a batch of jobs simultaneously, as long as the machine capacity is not exceeded. Only jobs of the same family can be grouped together in a batch. The batch processing time equals the longest processing time of all the jobs in the batch. The problem is to group the jobs into batches and then sequence the batches on the machines in such a way that the completion time of all batches is minimized (minimization of makespan). Since the problem belongs to the NP-Hard class, this work proposes: i) a Mixed Integer Programming model in order to obtain optimal solutions for small instances of the problem, and ii) two heuristic algorithms based on metaheuristics, in order to obtain high quality solutions in acceptable computational time, for instances of the large problem. The algorithms are based on Iterated Greedy - IG and Discrete Differential Evolution - DDE metaheuristics. Also, these metaheuristics are hybridized by combining them with local search methods, both algorithms present high quality and using a random selection of neighborhoods. The results of the experiments show that the performance of the proposed algorithms are of high quality, with the DDE-Hybrid algorithm showing the best results compared to the algorithms in the literature.

Keywords: Scheduling jobs. Parallel batch processing machines. Incompatibility of job family. Combination optimization. Heuristics. Metaheuristics.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

O sequenciamento de tarefas (*scheduling*) é um importante processo de tomada de decisão que ocorre em diferentes setores, especialmente em sistemas de manufatura. Problemas em lidar com a alocação de recursos para postos de trabalho durante determinados períodos de tempo e problemas de *scheduling* são problemas de otimização combinatória bastante estudados por terem grande importância prática e teórica (Jacoba and Arroyo, 2016). Nestes problemas, deseja-se encontrar uma solução de tal maneira que uma ou mais medidas de desempenho sejam otimizadas. Os problemas de *scheduling* ocorrem em muitas indústrias como a química, metalúrgica e têxtil, na de fabricação de semicondutores, na movimentação de carga no porto, transporte, sistemas de armazenamento e assim por diante (Jia et al., 2016). A maior parte destes problemas pertence à classe de problemas NP-difícil.

O problema de sequenciamento de tarefas em máquina de processamento em lote é conhecido como BPM (*Batch Processing Machine*). Uma BPM tem a capacidade de processar várias tarefas simultaneamente em um lote. As BPMs são encontradas em indústrias de peças discretas como componentes eletrônicos (Uzsoy, 1994; Cheng and Kovalyov, 1995), de fabricação de pneus (Oulamara et al., 2009) e em indústrias de aço (Tang and Liu, 2009; Tang et al., 2014). Este problema é motivado principalmente pela operação de gravação em semicondutores, em que circuitos integrados são colocados em um forno de alta temperatura, em lotes, por um longo período de tempo, para detectar falhas precoces, o que pode gerar um gargalo no processamento (Zhou et al., 2016; Lee et al., 1992; Uzsoy, 1994).

Este trabalho tem como foco estudar e propor métodos heurísticos para a resolução do problema de sequenciamento de família de tarefas em máquinas paralelas uniformes de processamento em lote, com capacidades não idênticas e tamanho de tarefas arbitrário.

1.1 O Problema e sua Importância

No processamento dos lotes, considera-se um conjunto de n tarefas, não idênticas pertencentes a F famílias, para serem processadas em um conjunto de m máquinas. Um lote é um conjunto de tarefas que podem ser agrupadas e processadas ao mesmo tempo. Um lote pode conter apenas tarefas de uma mesma família. O tempo de processamento de um lote é igual ao maior tempo de processamento entre todas as tarefas no lote. Esse tipo de operação se justifica por ser mais econômico processar um lote de tarefas do que processar cada tarefa individualmente. São consideradas máquinas paralelas uniformes aquelas que têm características distintas expressas em termos de velocidades de execução, pois cada máquina pode ter uma velocidade diferente de processamento, independentemente da tarefa que esteja sendo executada. O objetivo do problema é agrupar as tarefas em lotes, respeitando a capacidade da máquina, pois a soma do tamanho das tarefas não pode ultrapassar o tamanho da máquina, e, em seguida, determinar o sequenciamento desses lotes nas máquinas de tal maneira que seja minimizado o *makespan*, ou seja, o tempo de conclusão do último lote.

Vários pesquisadores têm tentado estender o problema de máquinas simples de processamento em lote para o ambiente das BPMs paralelas. Máquinas paralelas podem ser divididas em três classes: as idênticas, as não relacionadas e as uniformes. As BPMs idênticas são ambientes em que temos máquinas paralelas que não diferem entre si, em outras palavras, são consideradas tarefas que têm o mesmo tempo de processamento (p_j), qualquer das m máquinas que seja.

Segundo Pinedo e Hadavi (Pinedo and Hadavi, 1992), os problemas de sequenciamento podem ser descritos por uma notação com três campos: $\psi_1 \,|\psi_2 \,|\psi_3$. Nestes três campos $\psi_1 = \alpha m$, sendo m o número de máquinas, $\alpha = \{P, Q, R, F\}$ para máquinas, ψ_1 descreve o ambiente da máquina e contém apenas uma entrada. Este campo pode ter máquinas clássicas idênticas, uniformes, não relacionadas e *flow shop*, respectivamente. O campo ψ_2 fornece detalhes das características e restrições de processamento. Este campo pode conter nenhuma entrada, uma única ou várias. O campo ψ_3 descreve o objetivo a ser minimizado. Seguindo essa notação, o problema apresentado é denotado por $Qm \mid p_j, q_j, f_j, v_i, Q_i \mid C_{max}$. O campo Qm identifica as máquinas paralelas uniformes. No campo das restrições: p_j é o tempo de processamento; q_j, tamanhos; f_j, famílias diferentes para cada tarefa j; v_i, velocidade; e Q_i é a capacidade diferente em cada máquina. C_{max} denota que o objetivo é minimizar o *makespan*.

Heurísticas baseadas em FFLPT (*First Fit Longest Processing Time*) e BFLPT (*Best Fit Longest Processing Time*) (Damodaran and Chang, 2008) foram confrontadas com SA (*Simulated Annealing*) (Chang et al., 2004), tendo apresentado melhoria estatisticamente significativa. Chen (Chen et al., 2010) apresentaram duas heurísticas para

minimizar o *makespan*, GA (*Genetic Algorithm*) e ACO (*Ant Colony Optimisation*).

Damodaran e Vélez-Gallego trabalharam um problema $Pm \mid p_j, w_j, r_j \mid C_{max}$, propondo um SA e confrontando com MD (*Modified Delay*) e GRASP (*Greedy Randomized Adaptive Search Procedure*) (Damodaran and Vélez-Gallego, 2012). Chung *et al.* (Chung et al., 2009) estudaram o problema de minimizar o *makespan* considerando tamanhos de tarefas diferentes e tempos de liberação dinâmicos ($Pm \mid w_j, r_j \mid C_{max}$), propondo um modelo matemático e duas heurísticas. Damodaran *et al.* (Damodaran et al., 2011) consideraram o mesmo problema e apresentaram um GRASP. Wang e Chou (Wang and Chou, 2010) estudaram o problema de $Pm \mid r_j, w_j, q_j, Q_i, b \leq n \mid C_{max}$, propondo meta-heurísticas baseadas em SA e GA, superando os melhores resultados encontrados até então na literatura. As BPMs não relacionadas são ambientes mais generalizados, onde cada tarefa é processada em um tempo diferente, dependente da máquina, ou seja, p_{ji} tal que $j \in 1, ..., n$ e $i \in 1, ..., m$. As BPM Paralelas não relacionam tarefas com tamanhos iguais ($w_j = w$) e as BPMs com capacidades idênticas ($Q_i = Q$) têm sido pouco estudadas, sendo que na maioria das vezes são consideradas, além disso, poucos estudos consideram tempo de liberação (r_j).

Li *et al.* (Li et al., 2013b) consideraram o problema de BPMs não relacionadas, com tarefas de tamanhos não idênticos, apresentando um algoritmo de *Branch-and-bound*, também abordando o problema com heurísticas baseadas em BFLTP, apresentando sua contribuição para melhorias do método J_SC-BFLTP (*Job_shortest Completion Time BFLTP*). Arroyo e Leung (Arroyo and Leung, 2017b) consideraram o acréscimo de tempo de liberação das tarefas (r_j) no mesmo problema, aplicando heurísticas baseadas nas regras BF e FF .

Os autores citados acima propuseram um *Lower bound*, um modelo matemático e um algoritmo heurístico com base na meta-heurística IG (*Iterated greedy*) para o problema, considerando tempo de liberação e capacidade das máquinas não idênticas ($Rm \mid p_{ji}, s_j, r_j, Q_i \mid C_{max}$) (Arroyo and Leung, 2017a).

As BPMs uniformes são encontradas em ambientes em que as tarefas têm o mesmo tempo de processamento (p_j) em qualquer uma das m máquinas paralelas. Também consideramos nesse ambiente que as máquinas têm velocidades diferentes (v_i, tal que $i \in 1, ..., m$).

Xu e Bean (Xu and Bean, 2007) estudaram o problema para máquinas não idênticas, porém não foram consideradas velocidades, apenas que cada máquina tinha uma capacidade diferente e tarefas com tamanho distintos. Para solucionar, apresentaram um algoritmo genético com RKGA (*Random Keys Genetic Algorithm*). Damodaran *et al.* (Damodaran et al., 2012) propuseram um PSO (*Particle Swarm Optimization*) para solucionar o problema $Qm \mid p_j = v, Q_i \mid C_{max}$, mas não consideraram máquinas uniformes com diferentes velocidades.

Li *et al.* (Li et al., 2013a) apresentaram o problema $Qm \mid p_j, s_j, v_i \mid C_{max}$. Até onde

sabemos, estes autores foram os primeiros a considerar o problema de máquinas paralelas uniformes de processamento em lote. Eles propuseram heurísticas baseadas nas regras BF e BFLTP. Zhou *et al.* (Zhou et al., 2016) incluíram máquinas não idênticas, com capacidade diferente para cada máquina e tamanho de tarefa arbitrário, propondo um algoritmo evolutivo, chamado DDE (*Discrete Differential Evolution*), para problema ($Qm \mid p_j, s_j, v_i, Q_i \mid C_{max}$), comparando com resultados do modelo matemático os melhores algoritmos encontrados na literatura até então: o RKGA (Xu and Bean, 2007) e o PSO (Damodaran et al., 2012).

1.2 Motivação

O problema em questão é considerado de difícil solução, sendo caracterizado como um problema desafiador, por serem desconhecidos algoritmos eficientes para sua resolução. Assim, este trabalho se baseia na hipótese de que, para encontrar soluções de boa qualidade em um tempo computacional aceitável, devem ser utilizados algoritmos baseados em meta-heurísticas. Para tanto, são propostas adaptações das meta-heurísticas *Iterated Greedy* e *Discrete Differential Evolution*.

1.3 Objetivos

O objetivo geral deste trabalho é propor e desenvolver heurísticas baseadas em meta-heurísticas de busca em vizinhanças e em meta-heurísticas populacionais, que visem a agrupar as tarefas em lotes e encontrar uma sequência dos lotes nas máquinas paralelas para minimizar o tempo de conclusão do último lote (*makespan*), produzindo soluções de qualidade em tempo computacional aceitável. Além disso, propor um novo modelo de Programação Inteira-Mista para obter soluções ótimas para o problema.

1.3.1 Objetivos Específicos

Em busca de alcançar o objetivo geral, seguem alguns objetivos específicos a serem atingidos:

- Implementar a abordagem da literatura que trata o referido problema.
- Modelar e propor uma nova formulação matemática do problema.
- Propor novas abordagens heurísticas, baseadas em meta-heurísticas para este problema.

1.4 Estrutura do Trabalho

O restante deste trabalho está organizado da seguinte forma: O Capítulo 2 apresenta as características do problema em estudo neste trabalho. Também é apresentada a formulação do modelo matemático encontrado na literatura para o problema. O Capítulo 3 faz revisão da literatura sobre os trabalhos que abordam problemas em máquinas paralelas de sequenciamento de lotes e incompatibilidade de famílias de tarefas. No Capítulo 4, são expostas as meta-heurísticas propostas para solucionar o problema, enquanto no Capítulo 5, são mostrados os experimentos realizados neste trabalho. Por fim, o Capítulo 6 apresenta as conclusões deste trabalho.

Capítulo 2

Sequenciamento de Tarefas em Máquinas Paralelas Uniformes de Processamento em Lote

Neste capítulo, é definido o problema de sequenciar um conjunto de n tarefas com incompatibilidade de famílias em um conjunto de m máquinas paralelas de processamento em lotes uniformes e proposta uma modelagem de programação inteira mista. Nosso problema pode ser representado por $Q_m \mid p_j, s_j, b, v_i, Q_i, incompatible \mid C_{max}$. Na Seção 2.1, define o problema. Na Seção 2.2, são mostrados os parâmetros e variáveis de decisão. Um exemplo numérico é exibido na Seção 2.3. E, por fim, na Seção 2.4, é apresentado o Modelo Matemático.

2.1 Definição do Problema Abordado

O problema consiste em sequenciar n tarefas pertencentes a F famílias em máquinas paralelas uniformes de processamento em lote de modo a minimizar o *makespan*. Assim, o problema pode ser dividido em duas etapas principais: agrupar as tarefas em lotes, em seguida, alocar (sequenciar) esses lotes nas máquinas. Seguem os pressupostos básicos do problema:

- Uma tarefa pode ser processada somente em uma máquina, e cada máquina pode processar um trabalho de cada vez. Nenhuma preempção é permitida durante o processamento de uma tarefa.
- As tarefas precisam ser agrupadas em lotes antes de serem distribuídas às máquinas para processamento.
- Somente tarefas da mesma família podem ser agrupadas em um lote para processamento.

- As tarefas têm tamanhos e tempos de processamento não idênticos.
- As máquinas têm diferentes velocidades de processamento.
- O tamanho total dos lotes é limitado pela capacidade da máquina, variando de máquina para máquina.

2.2 Índices, Parâmetros e Variáveis de Decisão

Para uma melhor descrição do modelo, as notações são apresentadas da seguinte forma.

Notações e parâmetros do modelo:

- m número de máquina;
- n número de tarefa;
- F número de família;
- i índice da máquina, $i \in \{1, ..., m\}$;
- j índice da tarefa, $j \in \{1, ..., n\}$;
- b índice do lote, $b \in \{1, ..., n\}$;
- f índice da família, $f \in \{1, ..., F\}$;
- p_j tempo de processamento da tarefa j;
- q_j tamanho da tarefa j;
- Q_i capacidade da máquina i; e
- v_i velocidade da máquina i.

Variáveis de decisão:

- P_{bi} é o tempo de processamento do lote b na máquina i;
- x_{jbi} é igual a 1 se a tarefa j está programada no lote b na máquina i, caso contrário zero;
- y_{bif} é igual a 1 se as tarefas do lote b, na máquina i são da família f, caso contrário zero; e
- C_{max} é o *makespan*.

2.3 Exemplo Numérico

Considere uma instância do problema com $n = 9$ tarefas, $m = 2$ máquinas e $F = 3$ famílias. A Tabela 2.1 mostra o tempo de processamento (p_j), o tamanho (q_j) e a família (f_j) de cada tarefa j. Note que o tempo de processamento de cada tarefa j é o mesmo em cada máquina. A Tabela 2.2 mostra a capacidade e a velocidade de cada máquina i. As capacidades e as velocidades são diferentes.

Tabela 2.1: Dados das tarefas

j	1	2	3	4	5	6	7	8	9
p_j	5	4	5	4	3	6	7	2	8
q_j	3	2	3	3	1	3	2	2	1
f	1	2	3	2	3	3	1	1	2

Tabela 2.2: Dados das máquinas

i	1	2
Q_i	6	8
v_i	1,0	1,2

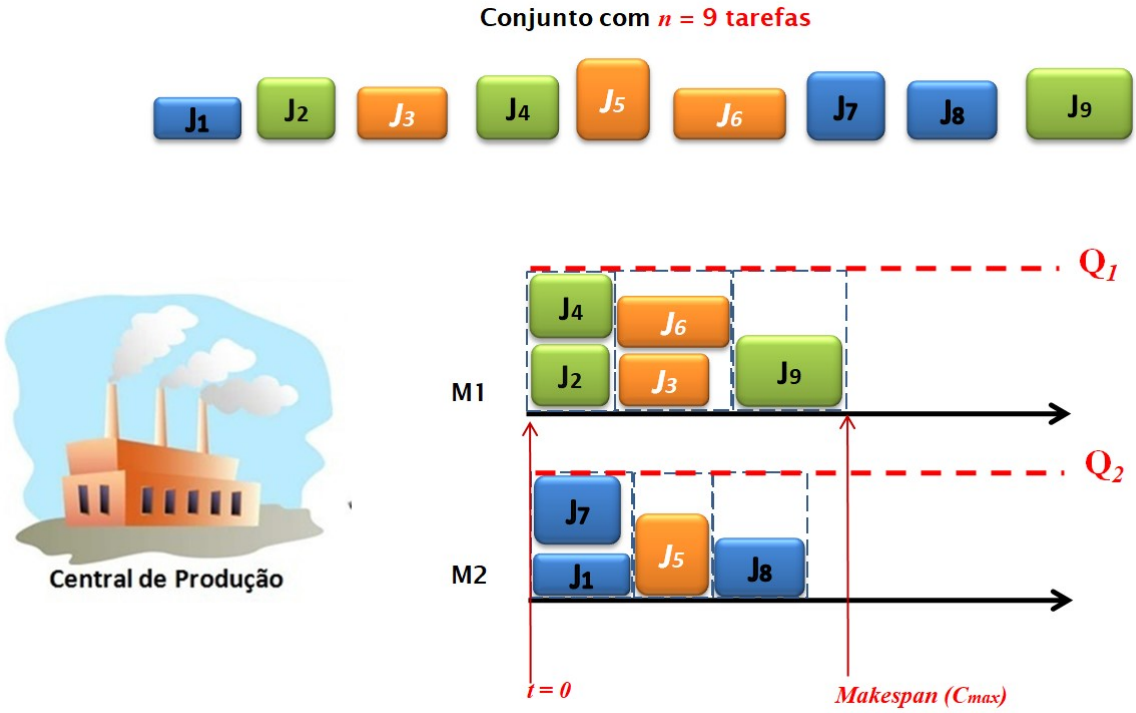


Figura 2.1: Sequenciamento de 9 tarefas em duas máquinas paralelas uniformes de processamento em lote.

O objetivo então é agrupar essas tarefas em lotes, em seguida, alocar esses lotes às máquinas para então serem processados e determinar sua sequência em cada má-

quina. Os lotes podem conter apenas tarefas de mesma família. A soma dos tamanhos das tarefas não pode ultrapassar a capacidade da máquina. A Figura 2.1 representa um exemplo de solução viável do problema para a instância apresentada. Note que as duas máquinas estão representadas no diagrama, bem como os lotes que foram agendados para cada máquina, com as tarefas que neles estão contidas, maiores que a máquina um $v_1 = 1,0$, $v_2 = 1,2$), ela processa tarefas com tempo, que são diferenciadas pelas cores de cada família. Como a máquina dois tem maior velocidade, seu processamento é maior em um menor tempo. O tempo de conclusão é apresentado no diagrama. Neste exemplo, o valor do *makespan* é o tempo de conclusão do terceiro lote na primeira máquina.

As máquinas 1 e 2 processam lotes de tarefas da mesma família.

2.4 Modelo Matemático

O modelo PIM é apresentado a seguir:

$$\min C_{max} \tag{2.1}$$

Sujeito a:

$$\sum_{b=1}^{n}\sum_{i=1}^{m} x_{jbi} = 1, \forall j = 1, ..., n \tag{2.2}$$

$$\sum_{j=1}^{n} q_j x_{jbi} \leq Q_i, \forall b = 1, ..., n; \forall i = 1, ..., m \tag{2.3}$$

$$P_{bi} \geq (p_j x_{jbi})/v_i, \forall j = 1, ..., n; \forall b = 1, ..., n; \forall i = 1, ..., m \tag{2.4}$$

$$\sum_{f=1}^{F} y_{bif} = 1, \forall b = 1, ..., n; i = 1, ..., m; \tag{2.5}$$

$$y_{bif} - x_{jbi} \geq 0, \forall b = 1, ..., n; i = 1, ..., m \tag{2.6}$$

$$C_{max} \geq \sum_{b=1}^{n} P_{bi}, \forall i = 1, ..., m; \tag{2.7}$$

$$x_{jbi} \in \{0,1\}, y_{bif} \in \{0,1\}, P_{bi} \geq 0, \forall j, b, i, f \tag{2.8}$$

A função objetivo (2.1) busca minimizar o *makespan*. A restrição (2.2) garante que todas as tarefas sejam alocadas e processadas em apenas um lote e em uma máquina. A restrição (2.3) assegura que a soma do tamanho de todas as tarefas do lote não

ultrapasse a capacidade da máquina. A restrição (2.4) determina o tempo de processamento do lote b na máquina i. As restrições (2.5) e (2.6) garantem que apenas trabalhos da mesma família possam ser agrupados. A restrição (2.7) determina o *makespan*. A restrição (2.8) determina as condições de domínio das variáveis de decisão.

Capítulo 3

Revisão de Literatura

Neste capítulo será feita uma revisão da literatura, em que serão apresentados os trabalhos mais relevantes que abordam máquinas paralelas de processamento em lote, considerando as possíveis características das máquinas e tarefas com incompatibilidade de famílias.

3.1 Máquinas de Processamento em Lote - BPM

Foram publicadas diversas pesquisas que consideram o problema de máquina de processamento em lote e de máquinas paralelas de processamento em lote, sendo a(s) máquina(s) idênticas (Potts and Kovalyov, 2000; Mathirajan and Sivakumar, 2006).

Já os problemas de sequenciamento de BPM paralelas não relacionadas foram muito menos estudados. Poucas pesquisas consideram tarefas com tamanhos diferentes e máquinas não idênticas. Esta revisão será dividida em duas partes - BPM paralelas uniformes e idênticas e BPM paralelas não relacionadas - em especial focando nos trabalhos em que as tarefas têm tamanho arbitrário e tempo de processamento distinto.

Vários estudiosos têm tentado estender o problema de *scheduling* BPM única ao ambiente BPM Paralelas. Desta forma, esta revisão abordará apenas problemas de BPM Paralelas.

3.1.1 BPM Idênticas

As BPMs idênticas são ambientes onde temos máquinas paralelas que não diferem entre si. Em outras palavras, são consideradas tarefas que têm o mesmo tempo de processamento (p_j) em qualquer das m máquinas que seja. Chang e Damodaran (Chang et al., 2004) sugeriram uma abordagem SA (*Simulated Annealing*) para o problema. O algoritmo superou a maioria das soluções encontradas pelo CPLEX. Mais tarde, Damodaran e Chang (Damodaran and Chang, 2008) propuseram dois

algoritmos heurísticos baseadas em BFLPT (*Best Fit Longest Processing Time*) e FFLPT (*First Fit Longest Processing Time*), que apresentaram uma melhoria estatisticamente significativa sobre o SA.

Chen *et al.* (Chen et al., 2010) apresentaram duas meta-heurísticas populacionais - o GA (*Genetic Algorithm*) e ACO (*Ant Colony Optimisation*) - para minimizar o *makespan*. Eles utilizaram uma regra heurística ERT-LPT para atribuir os lotes às máquinas paralelas, tendo concluído que o desempenho da GA foi melhor para instâncias pequenas do problema, enquanto a ACO foi melhor para as instâncias grandes do problema.

Damodaran e Vélez-Gallego (Damodaran and Vélez-Gallego, 2012) trabalharam com a variação do problema $Pm \mid p_j, w_j, r_j \mid C_{max}$, propondo um SA para minimizar o *makespan*, e confrontaram sua versão do SA com o modelo matemático e duas heurísticas: a MD (*Modified Delay*) e o GRASP (*Greedy Randomized Adaptive Search Procedure*). Os experimentos mostraram que as soluções encontradas pelos algoritmos SA e GRASP são compatíveis em relação à qualidade da solução em tempo computacional, além de ambas serem melhor que a heurística MD.

Chung *et al.* (Chung et al., 2009) estudaram o problema de minimizar o *makespan* considerando tamanhos de tarefas diferentes e tempos de liberação dinâmicos ($Pm \mid w_j, r_j \mid C_{max}$), propondo um modelo de programação linear inteira mista (MILP) e três algoritmos heurísticas eficientes.

Damodaran *et al.* (Damodaran et al., 2011) consideraram o mesmo problema e apresentaram um GRASP. Wang e Chou (Wang and Chou, 2010) estudaram o problema de $Pm \mid r_j, w_j, q_j, Q_i, b \leq n \mid C_{max}$, propondo meta-heurísticas baseadas em SA e GA, superando os melhores resultados encontrados até então na literatura.

3.1.2 BPM Uniformes

As BPMs uniformes são parecidas com ambientes com BPMs idênticas, em que temos tarefas com tempo de processamento (p_j) igual em qualquer das m máquinas, porém também consideramos que as máquinas têm velocidades e capacidades diferentes (v_i e Q_i tal que $i \in 1, ..., m$).

Xu e Bean (Xu and Bean, 2007) estudaram o problema para máquinas não idênticas, tendo levado em consideração apenas a capacidade individual de cada máquina, sem atenção voltada à velocidade e a tarefas com tamanhos distintos. Para solucionar, apresentaram um algoritmo genético com RKGA (*Random Keys Genetic Algorithm*) para solucionar o problema $Qm \mid p_j = v, Q_i \mid C_{max}$. Damodaran *et al.* (Damodaran et al., 2012) propuseram um PSO, mas não consideraram máquinas uniformes com diferentes velocidades.

Li *et al.* (Li et al., 2013a) apresentaram o problema $Qm \mid p_j, s_j, v_i \mid C_{max}$. Até onde sabemos, eles foram os primeiros a considerar o problema de máquinas paralelas

uniformes de processamento em lote, tendo proposto heurísticas baseadas nas regras BF e BFLTP.

Zhou *et al.* (Zhou et al., 2016) incluíram máquinas não idênticas, com capacidade diferente para cada máquina e tamanho de tarefa arbitrário, propondo um algoritmo evolutivo, chamado DDE (*Discrete Differential Evolution*), para problema $(Qm \mid p_j, s_j, v_i, Q_i \mid C_{max})$, comparando com resultados do modelo matemático e com os melhores algoritmos encontrados na literatura até então: o RKGA (Xu and Bean, 2007) e o PSO (Damodaran et al., 2012).

3.1.3 BPM Não Relacionadas

As BPMs não relacionadas são ambientes mais generalizados de máquinas idênticas e próximos da realidade, pois cada tarefa é processada em um tempo diferente dependente da máquina, ou seja, p_{ji} tal que $j \in 1, ..., n$ e $i \in 1, ..., m$. Problemas de *Scheduling* BPM Paralelas não relacionadas têm sido pouco estudados. Na maioria das vezes, as pesquisas focam nas tarefas com tamanhos iguais ($w_j = w$), BPMs com capacidades idênticas ($Q_i = Q$), além disso, poucas consideram o tempo de liberação (r_j).

Li *et al.* (Li et al., 2013b) consideraram o problema de BPMs não relacionadas, com tarefas de tamanhos não idênticos, apresentando um algoritmo de *Lower bound*, abordando através de heurísticas baseadas em BFLTP, apresentando sua contribuição para melhorias do método J_SC-BFLTP (*Job_shortest Completion Time BFLTP*). Arroyo e Leung (Arroyo and Leung, 2017b) consideraram para o mesmo problema o tempo liberação das tarefas (r_j), aplicando heurísticas baseadas nas regras BF e FF. Este autores propuseram dois algoritmos exatos, sendo um *Lower bound* e um modelo matemático e um algoritmo heurístico baseado na meta-heurística IG (*Iterated greedy*) para o problema, considerando o tempo de liberação e a capacidade das máquinas não idênticas $(Rm \mid p_j i, s_j, r_j, Q_i \mid C_{max})$ (Arroyo and Leung, 2017a).

3.2 Incompatibilidade de Famílias de Tarefas

Uma família de tarefas é um subconjunto de tarefas que podem ser processadas juntas em um mesmo lote, ou seja, tarefas de famílias diferentes não podem ser atribuídas ao um mesmo lote, pois cada família tem requisitos diferentes nas condições de processamento. Problemas de *scheduling* em uma única máquina BPM são mais comuns de serem encontradas onde as tarefas são divididas em famílias incompatíveis. Na prática, os clientes têm suas demandas sobre os produtos, tornando-se um desafio para as empresas otimizar a produção (Cheng et al., 2014).

Métodos exatos foram estudados para o problema de sequenciamento de tarefas

com incompatibilidade de famílias por Yuan *et al.* (Yuan et al., 2005), Fu *et al.* (Fu et al., 2009) e Liu *et al.* (Liu et al., 2010). Meta-heurísticas foram estudadas por Malve e Uzsoy (Malve and Uzsoy, 2007a), Chiang *et al.* (Chiang et al., 2010) e Tian *et al.* (Tian et al., 2011). Uzsoy (Uzsoy, 1995) considerou tarefas com tamanho iguais para otimizar os objetivos C_{max}, L_{max}, $\sum w_i C_i$; Mehta e Uzsoy (Mehta and Uzsoy, 1998), para otimizar o objetivo $\sum T_i$. Kempf *et al.* (Kempf et al., 1998) consideraram tarefas com tamanhos diferentes com o objetivo de minimizar $\sum C_i$ e C_{max} e Dobson e Nambimadom (Dobson and Nambimadom, 2001), para minimizar $\sum w_i C_i$.

Poucas pesquisas com máquinas paralelas de processamento em lote e incompatibilidade de tarefas foram desenvolvidas até onde sabemos. Malve e Uzsoy (Malve and Uzsoy, 2007b) consideraram, além de famílias e BPM paralelas, o tempo de liberação dinâmica das tarefas. Então propuseram uma heurística baseada em Algoritmo Genético e chaves aleatórias para minimizar o atraso máximo, mostrando-se consistente e eficiente. Mönch *et al.* (Mönch et al., 2005), buscando minimizar o atraso ponderado, propuseram duas abordagens diferentes de decomposição, em que a primeira abordagem forma lotes fixos, em seguida, atribuíram esses lotes às máquinas usando um algoritmo genético, finalmente, sequenciaram os lotes em máquinas individuais.

Capítulo 4

Meta-heurísticas propostas para o Problema

Nesta capítulo são apresentados os algoritmos heurísticos propostos para resolver o problema $Q_m \mid p_j, s_j, b, v_i, Q_i, incompatible \mid C_{max}$. Ele está dividido em seções. A seção 4.1 define os passos do algoritmo *Iterated Greedy* – IG. Na seção 4.2 o algoritmo *Iterated Greedy* Com Busca Local – IG-LS é diferenciado do algoritmo IG da seção 4.1. Por último na seção 4.3 apresentamos a Heurística *Discrete Differential Evolution* Híbrida - DDE-H.

4.1 Heurística *Iterated Greedy*

O *Iterated Greedy* – IG é um algoritmo meta-heurístico simples, mas que apresenta excelentes resultados, tendo sido criado para resolver problema de otimização combinatória, solucionando problemas de *scheduling* em ambiente *flow shop* (Ruiz and Stützle, 2007). O Algoritmo 1 apresenta o código do IG.

Algoritmo 1: Iterated Greedy (d, *Critrio_de_Parada*)

```
1  S ← construção inicial
2  S_B ← S
3  repita
4  |  S_P ← Destruição_Construção(S, d);
5  |  se f(S_P) < f(S) então
6  |  |  S ← S_P
7  |  |  se f(S) < f(S_B) então
8  |  |  |  S_B ←S;
9  |  |  fim
10 |  fim
11 até Critrio_de_Parada;
12 retorna S_B
```

O algoritmo IG começa a partir de uma solução inicial, após isso, é feita uma busca local nessa solução. Assim, tentativas de melhoramento na solução corrente acontecem em um ciclo iterativo, dividido em duas fases principais: destruição-construção e critério de aceitação. Durante a fase de destruição, são escolhidas aleatoriamente d tarefas a partir da solução atual para serem removidas, sendo estas tarefas armazenadas em um conjunto contendo apenas as tarefas removidas. Na fase de construção, essas tarefas removidas são inseridas através de um método guloso, onde são testadas todas as possíveis posições para uma tarefa j_k, sendo esta tarefa inserida em uma posição que gere um menor valor da função objetivo. Em seguida, a próxima tarefa (j_{k+1}) corresponde a todos os lotes possíveis, desde que não gere uma solução inválida, procurando qual tarefa repete os passos ($k \in \{1, .., d\}$). Se a solução encontrada for melhor que a melhor solução corrente (S), então esta solução é atualizada. Em seguida, uma segunda avaliação decide se a melhor solução conhecida (S_B) também será atualizada.

4.1.1 Representação da Solução

Uma solução é representada por uma sequência de tarefas que determinam a ordem em que elas serão dispostas nos lotes e nas máquinas através do algoritmo de formulação de lotes.

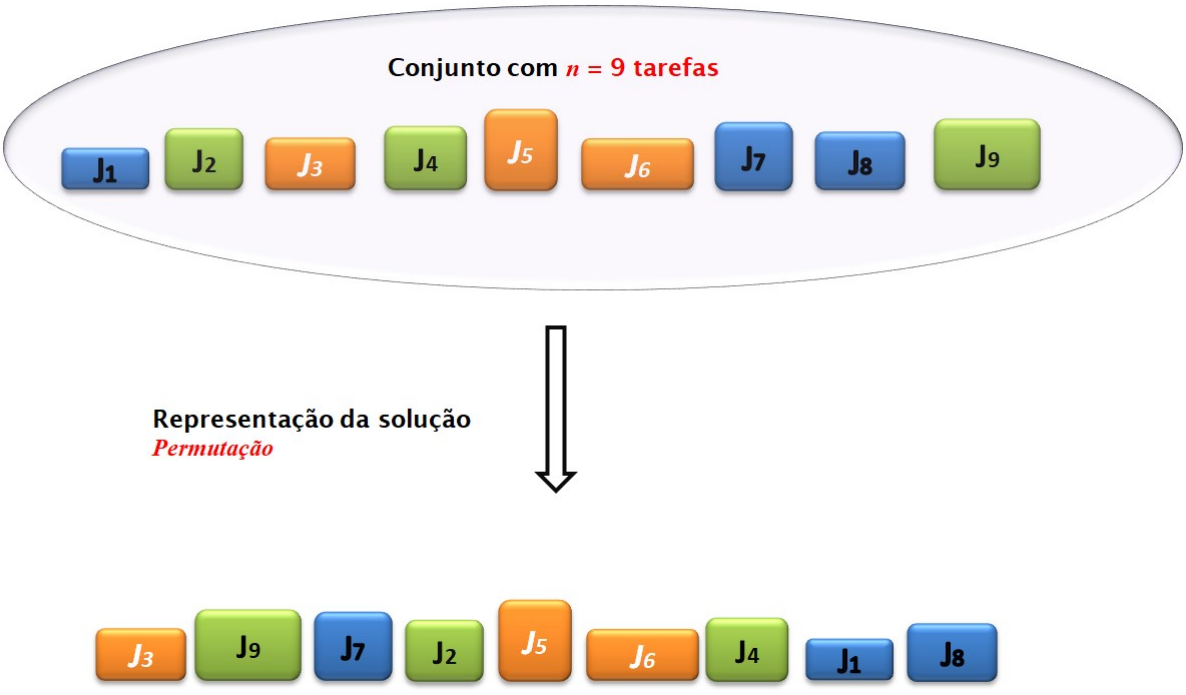


Figura 4.1: Conjunto com 9 tarefas e uma possível permutação representando uma solução.

A Figura 4.1 apresenta um conjunto com $n = 9$ tarefas, no qual podemos retirar

uma solução representada pela permutação $P = \{3,9,7,2,5,6,4,1,8\}$.

4.1.2 Construção Inicial

A solução inicial consiste em dois passos gulosos. Primeiramente, as tarefas são ordena- das por ordem decrescente da razão do tempo de processamento por tamanho (p_j/q_j). Ordenadas as tarefas, o segundo passo então é a formulação dos lotes. Na formulação dos lotes, é aplicado o algoritmo (*Batch formation and scheduling*) sugerido por Zhou *et al.* (Zhou et al., 2016).

Nesta etapa, as decisões são tomadas em conjunto, sendo a formulação dos lotes e seu agendamento nas máquinas dados por uma permutação das tarefas, pois são interdependentes. Para solucionar, utilizamos um método guloso, seguindo as seguintes etapas:

- Passo 0: Dada uma permutação de trabalho completa.
- Passo 1: Selecionar a primeira máquina k disponível. Se houver mais de uma máquina disponível, selecionar a com menor tempo de conclusão, e se persistir o empate, decidir por aquela com maior velocidade de processamento v_i. Crie um novo lote b, insira a tarefa e coloque o lote na máquina selecionada.

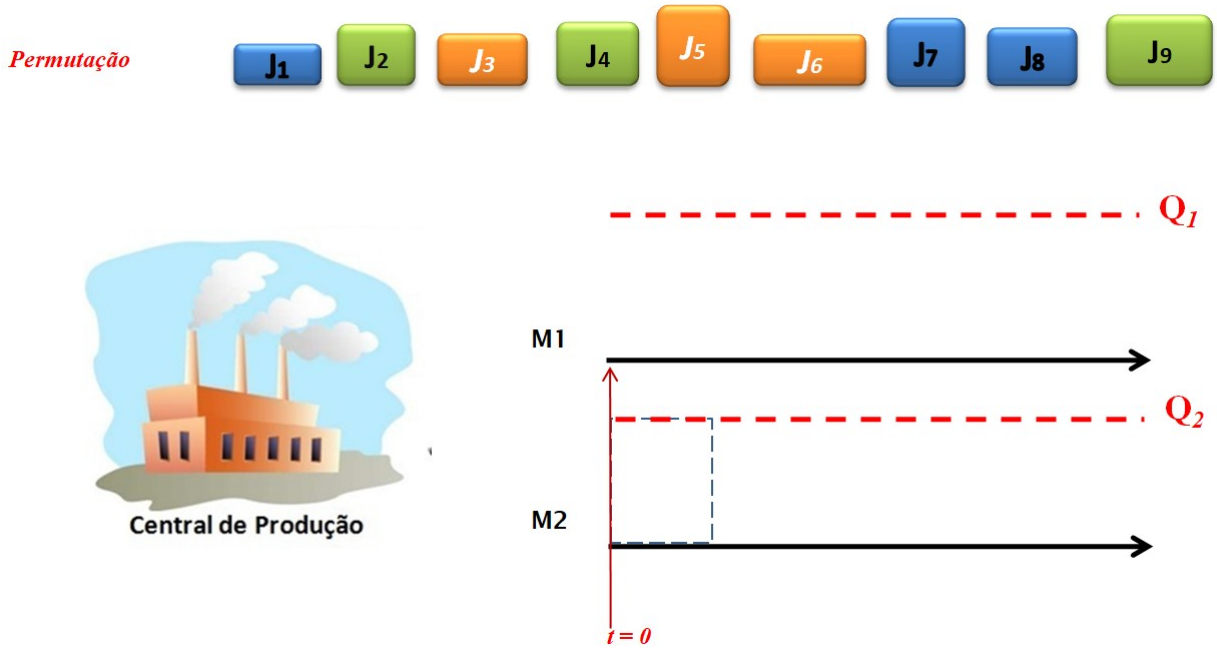


Figura 4.2: Representação da solução parcial após o Passo 1 do algoritmo de formulação dos lotes.

- Passo 2: Selecionar as tarefas à frente na permutação (j_{i+1}) para adicioná-la no lote b. Verifique se esta tarefa pode ser adicionada ao lote, respeitando a família

do lote e a capacidade da máquina. Se possível, adicione j_{i+1} ao lote b. Repita o 2º Passo até que o lote não tenha mais capacidade residual no lote, ou tenha percorrido todas as tarefas na permutação dada.

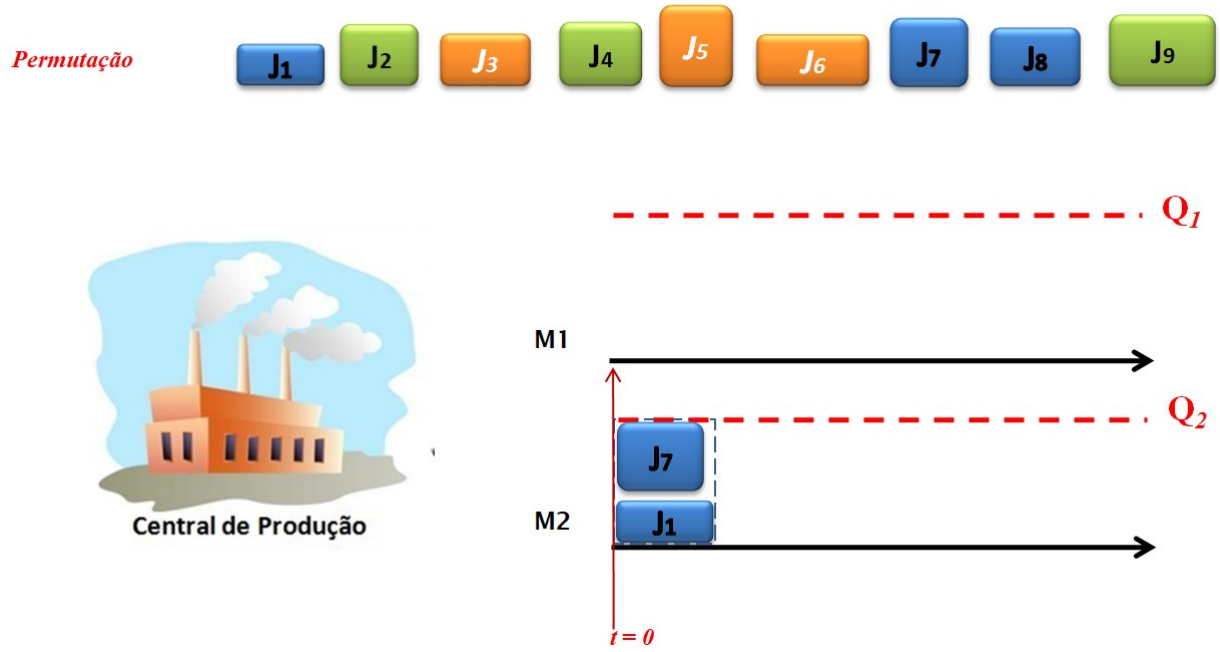


Figura 4.3: Representação da solução parcial após o Passo 2 do algoritmo de formulação dos lotes.

- Passo 3: Repita os passos 1 e 2 até que todas as tarefas da permutação estejam inseridas nos lotes e alocadas nas respectivas máquinas.

4.1.3 Destruição e Construção

Como mostrado acima, o IG é constituído de uma parte de destruição e construção da solução. O processo de destruição é realizado de modo iterativo, sendo a remoção de uma tarefa feita de cada vez a partir da solução atual, tendo como parâmetros o grau de destruição (d) e a solução (S_p) para ser perturbada. O parâmetro d define quantas destruições devem ser feitas na solução, ou seja, são escolhidas aleatoriamente d tarefas para serem removidas da solução inicial. Isso gera uma solução parcial (S_p) contendo apenas as tarefas que não foram removidas. As tarefas removidas são armazenadas em um conjunto auxiliar (J_r), onde são mantidas na ordem em que forem removidas.

A Figura 4.6 apresenta um exemplo com o parâmetro $d = 2$, em que cada iteração de uma tarefa é escolhida de forma aleatória, removida e inserida em um conjunto denominado J_r. Após as d iterações, temos a solução perturbada sem as tarefas que foram removidas e inseridas no conjunto J_r.

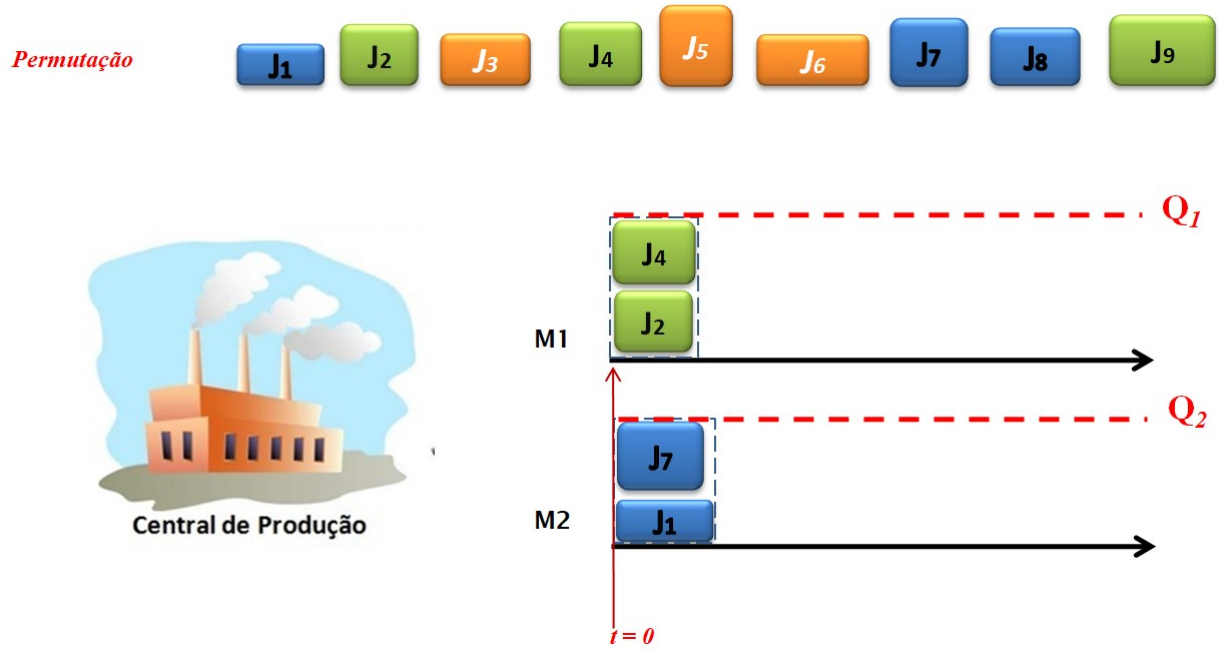


Figura 4.4: Representação da solução parcial após outra iteração do algoritmo (Repetindo os Passos 1 e 2).

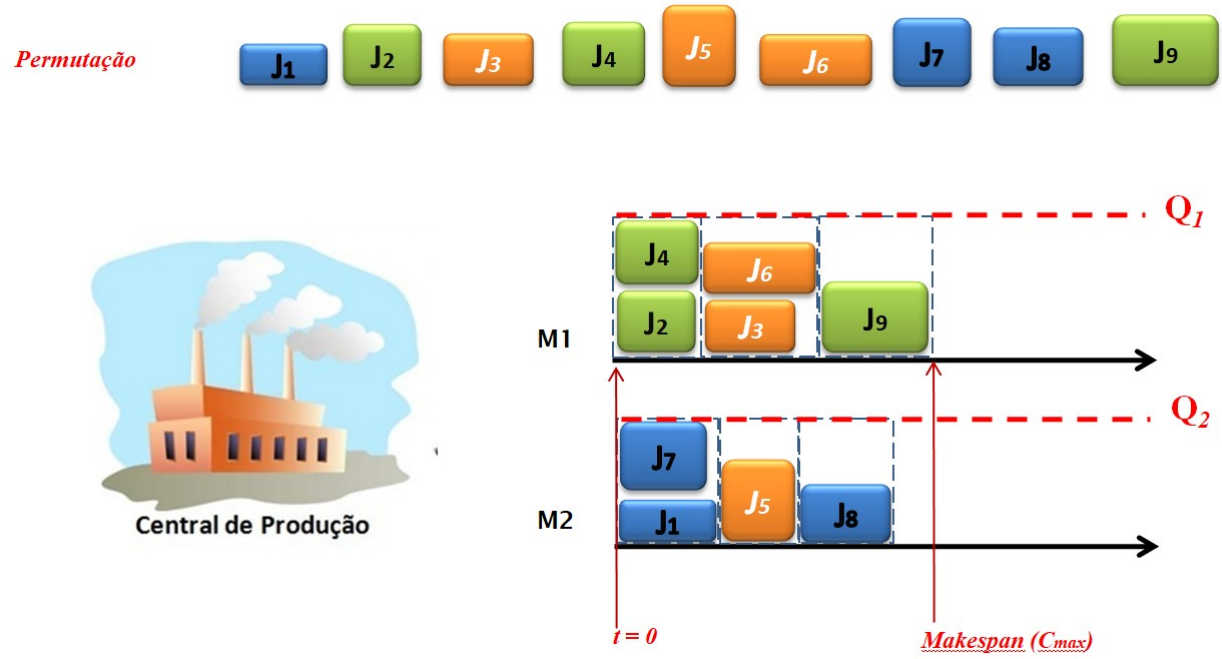


Figura 4.5: Representação da solução final após execução do algoritmo.

Após finalizada a destruição, é aplicado o algoritmo de formulação de lotes (Seção 4.1.2) e recalculados os tempos de conclusão desta solução parcial. O procedimento de construção aplica uma heurística construtiva gulosa para reconstruir uma solução completa, inserindo as tarefas removidas anteriormente. Este processo é feito inserindo cada tarefa em todas as posições possíveis na representação da solução, o que depende de procurar qual a melhor posição para a tarefa j armazenada no conjunto

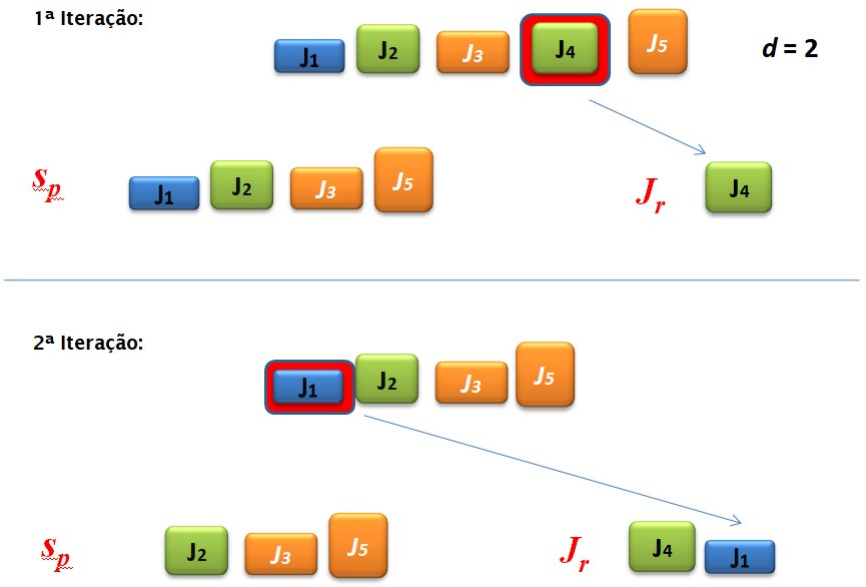


Figura 4.6: Etapa de destruição do algoritmo.

auxiliar J_r. Ele funciona da seguinte forma: a primeira tarefa de J_r é inserida em todos os possíveis posições viáveis existentes na representação da solução parcial S_p. Entre essas soluções parciais, a melhor solução entre elas (com o menor f) é escolhida pra continuar a próxima iteração. O processo se repete até que J_r esteja vazia, isto é, a fase da construção continua até que todas as tarefas removidas estejam reinseridas na solução parcial seguindo a mesma ordem em que elas foram retiradas, até obter uma solução completa com as n tarefas.

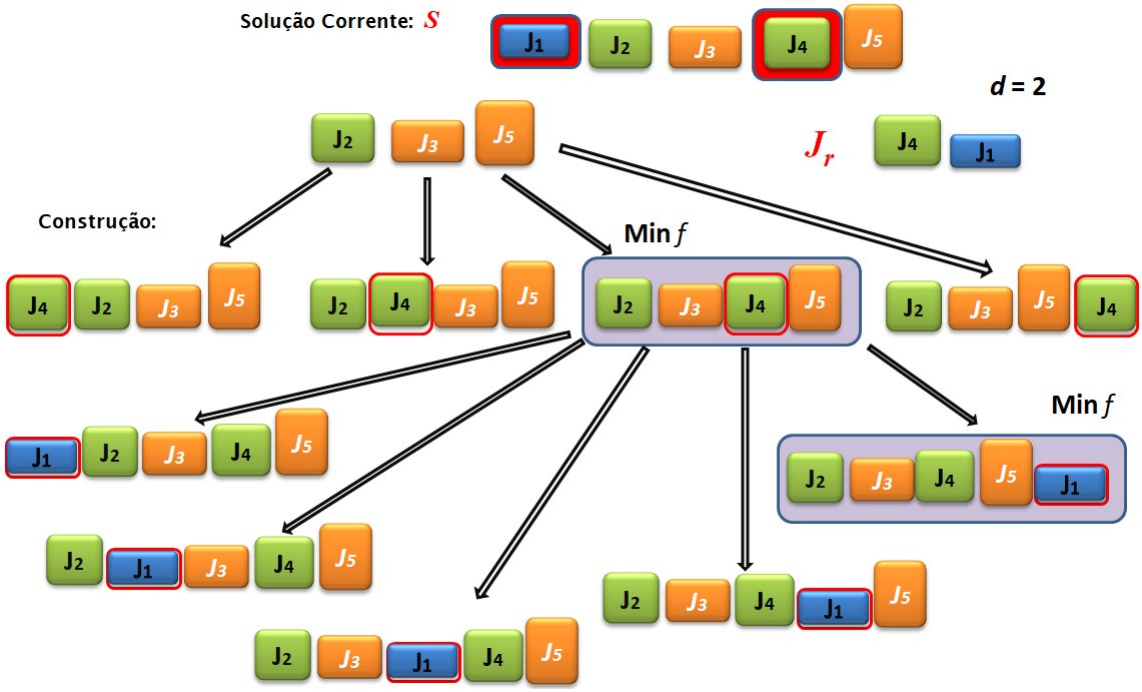


Figura 4.7: Etapa de construção do algoritmo.

A Figura 4.7 mostra a etapa de construção do algoritmo, onde as tarefas removidas que estavam no conjunto J_r são inseridas na mesma ordem de remoção na solução parcial S_p até que se tenha uma solução completa com as n tarefas. Neste exemplo saímos da solução corrente $S_c = \{1, 2, 3, 4, 5\}$ para a nova solução $S_n = \{2, 3, 4, 5, 1\}$.

4.1.4 Critério de Aceitação

Após as fases de destruição e construção, a nova solução é avaliada, sendo então decidido se esta nova solução será aceita. O critério de aceitação é um torneio simples. Se o $C_{max}(S_n)$ da nova solução for menor que $C_{max}(S_c)$ da solução corrente, a nova solução será aceita, atualizando, assim, a solução corrente.

4.2 Heurística *Iterated Greedy* Com Busca Local (IG-LS)

O *Iterated Greedy* com Busca Local – IG-LS inclui uma etapa de melhoria no algoritmo da Seção 4.1, acrescentando uma etapa de busca local. Esse método consiste em aplicar uma busca local após a fase de destruição-construção, buscando na vizinhança soluções melhores. O algoritmo 2 mostra o esboço do IG-LS com a modificação aplicada.

Algoritmo 2: Iterated Greedy with Local Search (d, *Critrio_de_Parada*)

```
1  S ← construção inicial
2  S_B ← S
3  repita
4  |  S_P ← Destruição_Construção(S, d);
5  |  Busca_Local(S_P);
6  |  se f(S_P) < f(S) então
7  |  |  S ← S_P
8  |  |  se f(S) < f(S_B) então
9  |  |  |  S_B ←S;
10 |  |  fim
11 |  fim
12 até Critrio_de_Parada;
13 retorna S_B
```

4.2.1 Busca Local

O método de busca local tem como objetivo melhorar a solução gerada pela fase de destruição e construção do IG. Este método procura por uma solução melhor em uma vizinhança. A vizinhança de uma solução contém todas as soluções alcançadas através de movimentos individuais em sua estrutura. Neste conjunto, a solução melhor do

que a solução atual é selecionada e o processo continua até que um ótimo local seja atingido. A estrutura de vizinhanças utilizadas na busca local é formada por soluções obtidas realizando trocas de tarefas na sequência da solução.

Duas tarefas são selecionadas aleatoriamente, e antes de realizar o movimento de troca, é verificado se a posição de destino de ambas as tem tarefas possui alguma tarefa de mesma família em posições adjacentes à sequência. Caso sim, esta é uma solução vizinha, o movimento é realizado e testada a nova solução. Essas trocas são feitas em uma porcentagem do número de tarefas (parâmetro $p\%$). O procedimento de busca local para se não houver melhoras para $p \cdot n$ consecutivas iterações.

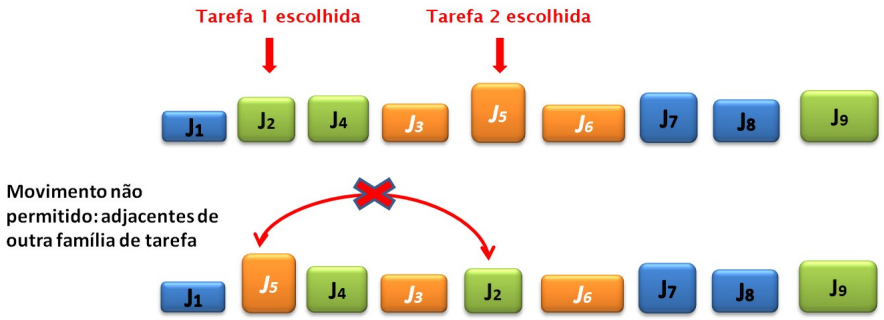


Figura 4.8: Exemplo 1 de movimento de vizinhança não permitido.

A Figura 4.8 mostra o exemplo em que duas tarefas são escolhidas aleatoriamente, mas que não podem executar o movimento de vizinhança, pois ambas não terão adjacentes da mesma família.

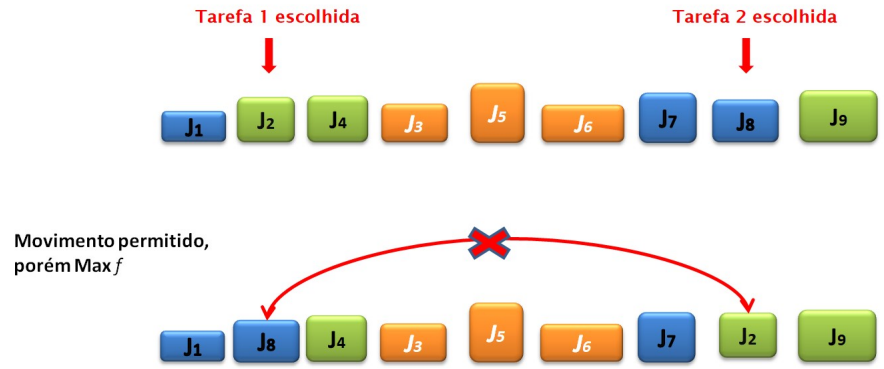


Figura 4.9: Exemplo 2 de movimento de vizinhança não permitido.

A Figura 4.9 mostra o exemplo em que pode ser executado o movimento de vizinhança, que gera uma solução vizinha pior (maior C_{max}).

A Figura 4.10 mostra o exemplo em que pode ser executado o movimento de vizinhança, e que gera uma solução vizinha melhor (menor C_{max}). Por consequência, o movimento é realizado e a solução corrente melhorada.

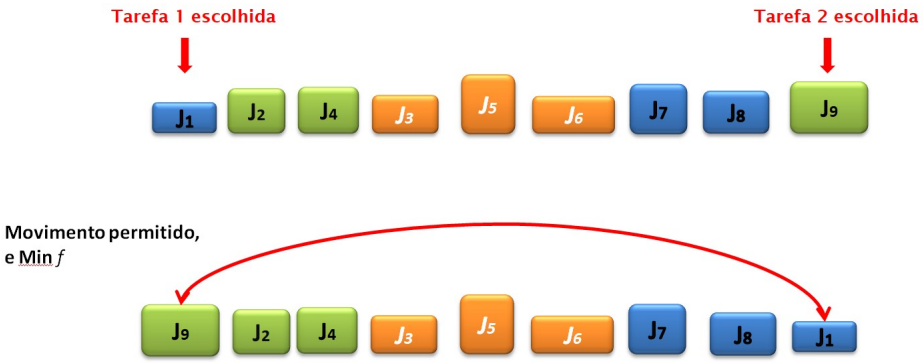


Figura 4.10: Exemplo de movimento de vizinhança permitido.

4.3 Heurística *Discrete Differential Evolution* Híbrida - DDE-H

O algoritmo DDE segue uma abordagem similar ao de um algoritmo evolucionário no qual são utilizados métodos de evolução de uma população, através de uma geração passada. Operadores de mutação são aplicados em alguns indivíduos pais, e operadores de *crossover* são aplicados para gerar novos indivíduos. Como em uma sociedade, há uma evolução natural dos indivíduos mais adaptados. O algoritmo 3 mostra o algoritmo do DDE-H.

Algoritmo 3: *Discrete Differential Evolution* (DDE)

```
Inicializa os parâmetros P, R, W, nE, k e d
t ← 0
Inicializa a população inicial com os P indivíduos X_i(t), i = 1,2,...,P
Inicializa a elite inicial com os nE indivíduos E_i(t), i = 1,2,...,nE
enquanto tempo < k * n(segundos) faça
    Realizar a mutação nos P indivíduos X_i(t) com E_i(t), gerando V_i(t) através do
     operador de mutação base.
    Aplicar o operador crossover nos P indivíduos V_i(t) indivíduos, gerando U_i(t).
    para i ← 1 até P faça
        Formule os lotes com as tarefas e agende os resultados nas máquinas pelo
         método apresentado na seção 4.1.2.
        Calcule a função objetivo.
    fim
    Determina os nE indivíduos da Elite
    Aplica a busca local nos nE indivíduos da Elite
    Determina os P indivíduos que farão parte da próxima geração, através da
     operação de seleção.
fim
retorna best_solution
```

4.3.1 Inicialização da População

Buscando uma população inicial mais diversificada e com alguma qualidade, a população inicial foi criada utilizando métodos de ordenação de tarefas: (1) em ordem decrescente de tempo; (2) em ordem decrescente de tamanho; (3) em ordem crescente do Tempo/Tamanho; (4) em ordem crescente por Tempo; (5) em ordem crescente por Tamanho; e (6) em ordem decrescente por Tempo/Tamanho. O restante da população foi inicializada ordenando as tarefas aleatoriamente sem nenhuma regra estabelecida. Em seguida, cada indivíduo desta população inicial passa pela formulação dos lotes, explicada a seguir.

4.3.2 Operador Discreto DE de Permutação de tarefa

Para obter um indivíduo mutante, um indivíduo da elite a e dois indivíduos da população anterior b e c são selecionados aleatoriamente, para então ser aplicado um operador de mutação baseado em permutação de trabalho, gerando então o indivíduo i da população atual, sendo a, b, c e i mutuamente diferentes ($a \neq b \neq c \neq i$, e $a, b, c, i \in \{1, 2, .., P\}$). Aplicando a estratégia de mutação *Differential Evolution* (DE) (Zhou et al., 2016), obtemos um novo indivíduo i. A Equação 4.1 representa este operador,

$$V_i(t) = X_a(t-1) \bigoplus R \bigotimes (X_b(t-1) - X_c(t-1)) \tag{4.1}$$

em que $R \in [0, 1]$ é o fator escalar de mutação, que controla a amplificação diferencial. O símbolo $\bigoplus$ é o operador OU Exclusivo (XOR), e o símbolo $\bigotimes$ é o operador tensor do produto. A Equação 4.2 mostra como a diferença ponderada entre dois indivíduos é feita

$$\Delta_i(t) = R \bigotimes (X_b(t-1) - X_c(t-1)) \tag{4.2}$$

e a Equação 4.3 mostra de forma detalhada como cada tarefa ($\delta_{ij}(t)$) do indivíduo $\Delta_i(t)$ é construída, pois $\Delta_i(t) = [\delta_{i1}(t), \delta_{i2}(t), .., \delta_{in}(t)]$ é o vetor temporário e r_j é o j^o é o valor aleatório uniformemente distribuído no intervalo $[0, 1]$.

$$\delta_{ij}(t) = \begin{cases} x_{bj}(t-1) - x_{cj}(t-1) & \text{se } r_j \leq R \\ 0 & \text{caso contrário} \end{cases} \tag{4.3}$$

Seguindo com a construção do indivíduo mutante $V_i(t)$, adicionamos ao $\Delta_i(t)$ o outro indivíduo $X_a(t-1)$ da população, como segue na Equação 4.4:

$$V_i(t) = X_a(t-1) \bigoplus \Delta_i(t) \tag{4.4}$$

mostrando de forma mais detalhada na Equação 4.5 como cada tarefa é construída, e considerando % como o operador módulo do resultado da divisão de um numerador por um denominador. O operador modulo é usado para garantir que cada tarefa de $V_i(t)$ representa um número legal, isto é, $1 \leq v_{ij}(t) \leq n$.

$$v_{ij}(t) = x_{aj}(t-1) \bigoplus \delta_{ij} = (x_{aj}(t-1) + \delta_{ij}(t) + n)\%n + 1 \tag{4.5}$$

Repare que $V_i(t)$ pode não representar uma permutação completa, pois pode acontecer de algumas tarefas aparecerem duplicadas, bem como de algumas tarefas nem serem selecionadas nesse vetor. Para solucionar as tarefas duplicadas, podemos desconsiderar as repetições. Já para completar com as demais tarefas que faltam na permutação, é aplicado um operador de recombinação, combinando o indivíduo mutante $V_i(t)$ com seu correspondente na Xi da população anterior $(t-1)$. Para isso denotamos o parâmetro de *crossover* como $W \in [0,1]$. A operação de *crossover* é dada pela Equação 4.6

$$u'_{ij}(t) = \begin{cases} v_{ij}(t) & \text{se } r_j \leq W \\ 0 & \text{caso contrário} \end{cases} \tag{4.6}$$

Basta produzir $U_i(t)$ preenchendo as posições vazias de $U'_i(t)$ com as tarefas remanescentes de $X_i(t-1)$ na sua ordem original. Desta forma, teremos uma solução completa com todas as tarefas ordenadas através das operações de permutação e *crossover*.

4.3.3 Formulação de Lotes e Agendamento nas Máquinas

Nesta etapa, segue o algoritmo que foi anteriormente explicado na Seção 4.1.2.

4.3.4 Seleção dos nE Indivíduos do Conjunto Elite e Busca local

Os nE melhores indivíduos desta geração são selecionados automaticamente para fazer parte do conjunto Elite. Estes indivíduos recebem então uma etapa de busca local para encontrar soluções vizinhas melhores. No processo de busca local é aplicado o algoritmo IG apresentado na Seção 4.1. A solução da elite é destruída e construída, e verificado o critério de aceitação. O critério de parada nesta caso é quando após uma iteração na busca não há melhoras na solução.

4.3.5 Seleção dos P Indivíduos para a Próxima Geração

Os indivíduos que farão parte da próxima geração $(X_i(t))$ são selecionados de duas formas. Os indivíduos do conjunto elite são selecionados automaticamente, e os

demais, seguindo a Equação 4.7. Primeiro o indivíduo ($U_i(t)$) é avaliado, e calculada sua solução (f). Se essa solução for melhor que a solução $X_i(t-1)$, então ela passa para a próxima geração. Caso contrário, mantemos a mesma solução da geração anterior.

$$X_i(t) = \begin{cases} U_i(t) & \text{se } f(U_i(t)) \leq f(X_i(t-1)) \\ X_i(t-1) & \text{caso contrário} \end{cases} \tag{4.7}$$

Capítulo 5

Experimentos Computacionais

Os experimentos computacionais foram divididos em etapas. A primeira etapa é mostrada na Seção 5.1, onde são apresentadas as instâncias dos problemas-teste. A Seção 5.2 apresenta a métrica utilizada para avaliar os algoritmos heurísticos. A etapa de calibração dos parâmetros utilizados nos algoritmos propostos é mostrada na Seção 5.3. Por fim, a Seção 6.1 é dividida em dois grupos. O grupo de Instâncias de Pequeno Porte onde são apresentados os resultados dos algoritmos e do Modelo Matemático, e o grupo Instância de Grande Porte, onde são apresentados os resultados experimentais e a análise dos algoritmos.

Os testes foram feitos em um computador Intel Core i7, 4GHz, com 32 GB de RAM, sistema operacional Ubuntu 18.04.3 LTS - 64 bits. Os algoritmos propostos foram implementados em C++ e executados com uma simples *thread*.

5.1 Geração das Instâncias

Zhou *et al.* (Zhou et al., 2016) identificaram que fatores como número de tarefas, número de máquinas, tamanho das tarefas, tempo de processamento variado e capacidade e velocidade das máquinas afetariam as instâncias. Sendo assim, ele produziu um grupo com instâncias aleatórias, seguindo níveis definidos para variação dos fatores (gerados aleatoriamente), para o problema $Qm \mid p_j, s_j, v_i, Q_i \mid C_{max}$. Essas instâncias foram adaptadas para o problema em estudo, acrescentando informações de famílias de tarefas, escolhidas por uma distribuição aleatória, nos intervalos definidos.

A Tabela 5.1 mostra os níveis definidos para variação dos fatores (gerados aleatoriamente). Para cada combinação de fatores e níveis, foram geradas 10 instâncias aleatórias do problema, e cada instância foi solucionada 5 vezes por cada algoritmo. Cada categoria de instâncias representa um código de execução. Por exemplo, a categoria de problema com 50 tarefas, 4 famílias, 2 máquinas e tamanho de tarefas geradas no

intervalo $[1, 20]$ é chamada por J2F2M2s1. Já categoria de problema com 100 tarefas, 12 famílias, 3 máquinas e tamanho de tarefas geradas no intervalo $[10, 30]$ é chamada por J3F5M3s2. Todas as instâncias utilizadas pelos autores foram disponibilizadas, adaptadas e utilizadas neste trabalho.

Tabela 5.1: Resumo das variações das instâncias

Fatores	Níveis
Número de tarefas, n	20, 50, 100, 200 e 300
Número de famílias, F	2 e 4, para $n \in \{20, 50\}$; 3, 6, 12 para $n \in \{100, 200, 300\}$
Tamanho das tarefas, s_j	Uniforme no intervalo [1, 20] e [10, 30]
Tempo de processamento das tarefas	Uniforme no intervalo [8, 48]
Número de máquinas, m	2, 3, 4 e 5
Capacidade da máquina, S_i	$\{40, 50, 60\}$
Velocidade da máquina, v_i	$\{1.0, 1.2, 1.4, 1.6, 1.8, 2.0\}$

As instâncias foram divididas em dois grupos, de Pequeno Porte e de Grande Porte. As instâncias de pequeno porte correspondem às instâncias que têm $n \in \{20, 50\}$ de número de tarefas e $F \in \{2, 4\}$ para o número de famílias. Este grupo conta com 320 instâncias e foi usado nos teste do Modelo Matemático. As instâncias de grande Porte foram geradas com o número de tarefas $n \in \{100, 200, 300\}$ e $F \in \{3, 6, 12\}$. Este grupo tem 720 instâncias.

5.2 Métrica de Avaliação

Para mensurar a qualidade de uma solução, tanto na calibração dos parâmetros das heurísticas, quanto nos experimentos de comparação, foi utilizada a medida do Desvio Percentual Relativo (RPD - *Relative Percentage Deviation*). O RPD é computado para uma dada instância, de acordo com a equação a seguir (Vallada et al., 2008):

$$RPD(\%) = 100 \times \frac{f_{algoritmo} - f_{melhor}}{f_{melhor}} \tag{5.1}$$

Em que $f_{algoritmo}$ corresponde ao valor da média *makespan* obtida por todas as execuções do algoritmo, e f_{melhor} corresponde à melhor solução *makespan* encontrada a partir da execução de todos os algoritmos comparados. O valor do RPD é considerado de grande qualidade, quanto menor for o seu valor.

Como não é viável a comparação com a solução ótima, a comparação mostrada na Seção 5.4.2 é feita considerando a melhor solução conhecida. Para as intâncias de pequeno porte, mais precisas com $n = 20$, foram obtidas soluções ótimas na maioria das instâncias, através da resolução do Modelo de Matemático no software IBM CPLEX versão 12.5. Esses resultados são apresentados na Seção 5.4.1.

5.3 Calibração dos Parâmetros dos Algoritmos Heurísticos

Para calibração dos parâmetros dos algoritmos, foi gerado um conjunto independente de 120 novas instâncias: 40 instâncias de pequeno porte e 80 instâncias de grande porte. A Tabela 5.2 mostra as combinações de fatores utilizadas para a geração aleatória das instância de calibração. Para cada combinação, foram geradas 5 instâncias.

Tabela 5.2: Resumo das variações das instâncias de calibração

Fatores	Níveis
Número de tarefas, n	$50, 100, 200$
Número de famílias, F	$2, 4$, para $n = 50$; $6, 12$ para $n \in \{100, 200\}$
Tamanho das tarefas, s_j	Uniforme no intervalo $[1, 20]$ e $[10, 30]$
Tempo de processamento das tarefas	Uniforme no intervalo $[8, 48]$
Número de máquinas, m	$3, 4$

Na calibração, os algoritmos foram executados 5 vezes para cada instância. Os resultados obtidos são analisados utilizando o RPD, o qual é calculado considerando a média do *makespan* das 5 execuções para cada uma das instâncias.

Alguns parâmetros são utilizados por todos os algoritmos. Portanto, por um critério de igualdade entre os algoritmos, estes parâmetros foram definidos de forma global para todos os algoritmos. Esses parâmetros são:

- *Critério de Parada*: Representa o tempo de execução para os algoritmos propostos, tendo sido estabelecido o tempo máximo de CPU pela expressão $k \cdot n$ segundos. Nesta expressão, o tempo do CPU é influenciado pelo número de tarefas (n). Para n fixado, o tempo de execução aumenta quando o valor de n cresce.

- *Critério de Aceitação*: A aceitação de uma solução é considerada apenas quando temos uma solução viável, sendo esta solução melhor que a solução corrente.

- k: O parâmetro k foi definido como $k = 0,30$ para os experimentos de calibração e $k = 0,75$ para os experimentos finais.

5.3.1 Calibração do Parâmetro do Algoritmo *Iterated Greedy*

Um experimento preliminar foi conduzido para definir os parâmetros para o algoritmo proposto IG. Existe um parâmetro que afeta a performance do algoritmo, que é o número de tarefas removidas na destruição da solução d, que define quantas tarefas serão retiradas da solução na fase de destruição. Foram considerados diferentes valores, sendo eles: $d \in \{1, 2, 4, 6\}$. A Figura 5.1 mostra o gráfico de médias resultante do

teste Tukey da Diferença Honestamente Significativa (HSD) como nível de confiança de 95%. Na Figura 5.1, pode-se ver que o parâmetro $d = 3$ produz as menores médias. Desta forma, o valor de $d = 3$ foi adotado na execução do algoritmo.

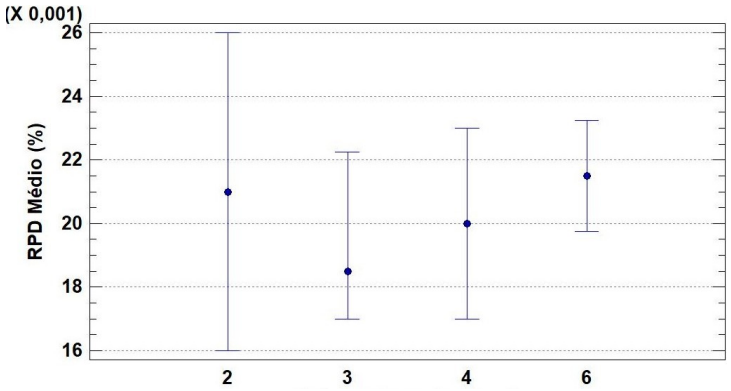


Figura 5.1: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração do parâmetro d

5.3.2 Calibração dos Parâmetros do Algoritmo *Iterated Greedy* com Busca Local

Os parâmetros para o algoritmo proposto IG com Busca Local analisados são d o número de tarefas removidas na destruição da solução $d \in \{1, 2, 4, 6\}$ e p porcentagem de tarefas escolhidas aleatoriamente para serem utilizadas na busca local $p \in \{10\%, 20\%, 40\%\}$.

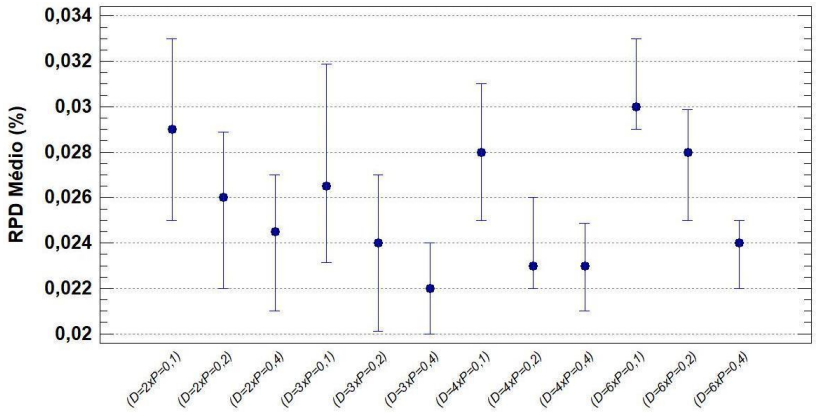


Figura 5.2: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração do parâmetros do IG-LS

A Figura 5.2 mostra o gráfico de médias resultante do teste Tukey da Diferença Honestamente Significativa (HSD) com nível de confiança de 95%. Na Figura 5.2, pode-se ver que os parâmetros $d = 3$ e $p = 40\%$ produzem as menores médias. Desta forma, o valores de $d = 3$ e $p = 40\%$ foram adotados na execução do algoritmo.

5.3.3 Calibração dos Parâmetros do Algoritmo DDE-Híbrido

Os parâmetros analisados são:

- R: Fator escalar de mutação.
- W: Parâmetro de *crossover*.
- k: Tempo de CPU ($k \cdot n$ segundos).
- d: Número de remoções na busca local.
- p: Tamanho da população.
- nE: Tamanho da elite.

Os valores testados para os parâmetros estão especificados na Tabela 5.3. Observa-se um total de 36 combinações de parâmetros do algoritmo DDE-H a serem avaliados. Os parâmetros R, W e k foram previamente fixados como mostrado na Tabela 5.3. Os valores de R e W seguem os valores do algoritmo de referência DDE.

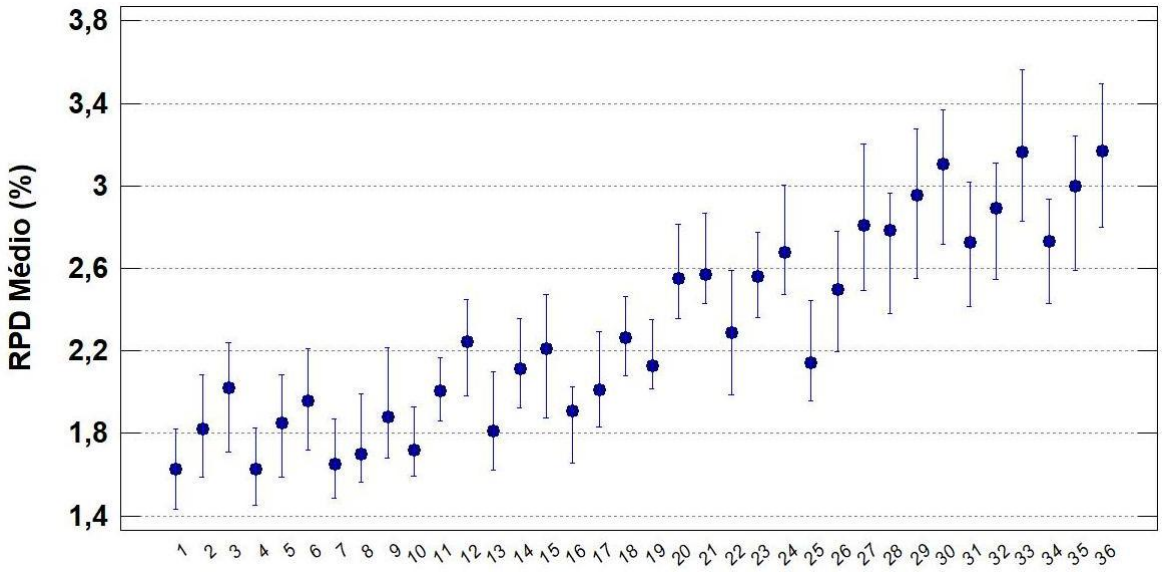


Figura 5.3: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração dos parâmetros do DDE-H

A Figura 5.3 mostra o gráfico de médias resultantes do teste Tukey da Diferença Honestamente Significativa (HSD) como nível de confiança de 95%. Na Figura 5.3, pode-se ver que as combinações 1 e 4 apresentam os melhores resultados. Portanto foram definidos como parâmetros os valores da combinação 4, pois o tamanho da população segue o mesmo tamanho do algoritmo de referência DDE.

Tabela 5.3: Combinações dos parâmetros do DDE-H

Combinação	Parâmetros					
	R	W	k	d	p	nE
1	0,5	0,1	0,3	2	30	2
2	0,5	0,1	0,3	2	30	4
3	0,5	0,1	0,3	2	30	6
4	0,5	0,1	0,3	2	40	2
5	0,5	0,1	0,3	2	40	4
6	0,5	0,1	0,3	2	40	6
7	0,5	0,1	0,3	2	50	2
8	0,5	0,1	0,3	2	50	4
9	0,5	0,1	0,3	2	50	6
10	0,5	0,1	0,3	4	30	2
11	0,5	0,1	0,3	4	30	4
12	0,5	0,1	0,3	4	30	6
13	0,5	0,1	0,3	4	40	2
14	0,5	0,1	0,3	4	40	4
15	0,5	0,1	0,3	4	40	6
16	0,5	0,1	0,3	4	50	2
17	0,5	0,1	0,3	4	50	4
18	0,5	0,1	0,3	4	50	6
19	0,5	0,1	0,3	6	30	2
20	0,5	0,1	0,3	6	30	4
21	0,5	0,1	0,3	6	30	6
22	0,5	0,1	0,3	6	40	2
23	0,5	0,1	0,3	6	40	4
24	0,5	0,1	0,3	6	40	6
25	0,5	0,1	0,3	6	50	2
26	0,5	0,1	0,3	6	50	4
27	0,5	0,1	0,3	6	50	6
28	0,5	0,1	0,3	8	30	2
29	0,5	0,1	0,3	8	30	4
30	0,5	0,1	0,3	8	30	6
31	0,5	0,1	0,3	8	40	2
32	0,5	0,1	0,3	8	40	4
33	0,5	0,1	0,3	8	40	6
34	0,5	0,1	0,3	8	50	2
35	0,5	0,1	0,3	8	50	4
36	0,5	0,1	0,3	8	50	6

5.4 Resultados Experimentais

Nesta seção, são apresentados os resultados obtidos pelos algoritmos propostos: IG, IG-LS e DDE-H bem como o algoritmo exato, apresentado na forma do modelo matemático e executado pelo CPLEX. Estes algoritmos são comparados entre si e também com o algoritmo encontrado no estado da arte da literatura. No nosso conhecimento, o melhor algoritmo proposto para um problema próximo do estudado neste trabalho, é o DDE que apresenta um desempenho muito bom para o problema $Qm \mid p_j, s_j, v_i, Q_i \mid C_{max}$, abordado por Zhou (Zhou et al., 2016). Esta seção está dividida em três subseções. A Seção 5.4.1 mostra os resultados obtidos pelos algoritmos heurísticos e modelo matemático para as instâncias de pequeno porte. A Seção

5.4.2 apresenta os resultados obtidos pelos algoritmos heurísticos para as instâncias de grande porte. E por fim, a Seção 5.4.3 faz a análise de convergência dos resultados dos algoritmos heurísticos propostos neste trabalho.

5.4.1 Resultados Experimentais para as Instâncias de Pequeno Porte

Nesta seção, é feita a comparação do desempenho do modelo matemático executado pelo CPLEX, com os resultados obtidos pelos algoritmos propostos neste trabalho: IG, IG-LS, DDE-H e pela versão implementada da literatura (DDE). Foram realizados experimentos com o grupo das instâncias de pequeno porte. Todos os algoritmos foram executados 5 vezes com o mesmo tempo de CPF ($k \cdot n$ segundos) como critério de parada. Já o algoritmo exato, teve o tempo de CPU limitado a 3600 segundos, caso não fosse encontrada a solução ótima. Os resultados obtidos são analisados utilizando o valor do RPD apresentado na Equação 5.1, o qual é calculado pela média do *makespan* em 5 execuções de cada instância, sendo o melhor valor encontrado entre todas as execuções de todos os algoritmos.

Tabela 5.4: Resultados RPDs para as instâncias com $n = 20$ tarefas

	CPLEX	IG		IG-LS		DDE		DDE-H	
Instâncias	F.O.	Mínimo	Médio	Mínimo	Médio	Mínimo	Médio	Mínimo	Médio
J1M2F1s1	**0,00**	0,64	1,19	0,64	0,64	0,64	0,64	0,64	0,64
J1M2F1s2	**0,06**	0,55	1,04	0,35	0,49	0,48	0,50	0,48	0,53
J1M2F2s1	**0,00**	0,40	0,54	0,08	0,34	0,08	0,08	0,08	0,11
J1M2F2s2	**0,00**	**0,00**	0,40	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
J1M3F1s1	**0,00**	3,14	4,14	2,75	2,93	2,75	2,93	2,97	3,01
J1M3F1s2	**0,00**	0,75	2,54	0,46	0,56	0,58	0,65	0,46	0,99
J1M3F2s1	**0,00**	**0,00**	0,11	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
J1M3F2s2	**0,00**	0,57	0,93	0,09	0,34	0,09	0,21	0,09	0,30
J1M4F1s1	**0,00**	1,84	2,22	**0,00**	0,40	**0,00**	0,55	**0,00**	0,55
J1M4F1s2	**0,00**	1,38	2,78	0,24	0,35	0,12	0,73	0,66	1,47
J1M4F2s1	**0,00**	0,41	0,41	0,41	0,41	0,41	0,41	0,41	0,41
J1M4F2s2	**0,00**	0,13	1,55	**0,00**	0,23	**0,00**	0,09	**0,00**	0,30
J1M5F1s1	**0,00**	**0,00**	0,36	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
J1M5F1s2	**0,36**	0,63	2,73	0,17	0,73	0,43	0,84	0,32	1,26
J1M5F2s1	**0,00**	**0,00**	0,13	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
J1M5F2s2	**0,15**	0,28	1,67	**0,00**	0,27	**0,00**	0,20	**0,00**	0,59
Média	**0,04**	0,67	1,42	0,33	0,48	0,35	0,49	0,38	0,64

A Tabela 5.4 mostra os valores médios dos RPDs mínimos e RPDs médios calculados a partir dos resultados médios (das 5 execuções), encontrados pelos algoritmos comparados para todas as instâncias de pequeno porte, com $n = 20$ tarefas. Os valores médios de RPDs são agrupadas pelo combinação de tarefas (n), máquinas (m), família (F) e tamanho (s). Nesta tabela, estão exibidos também na primeira coluna os resultados do RPD do CPLEX para cada conjunto. Podemos perceber que o CPLEX encontrou soluções ótimas para 13 conjuntos, com média global de RPDs igual a $0,04\%$ é assim o melhor algoritmo para os conjuntos com $n = 20$. Os algoritmos IG-

LS, DDE, DDE-H e IG quem têm a média global de RPDs igual 0,48%, 0,4%, 0,64%, 1,42%, respectivamente.

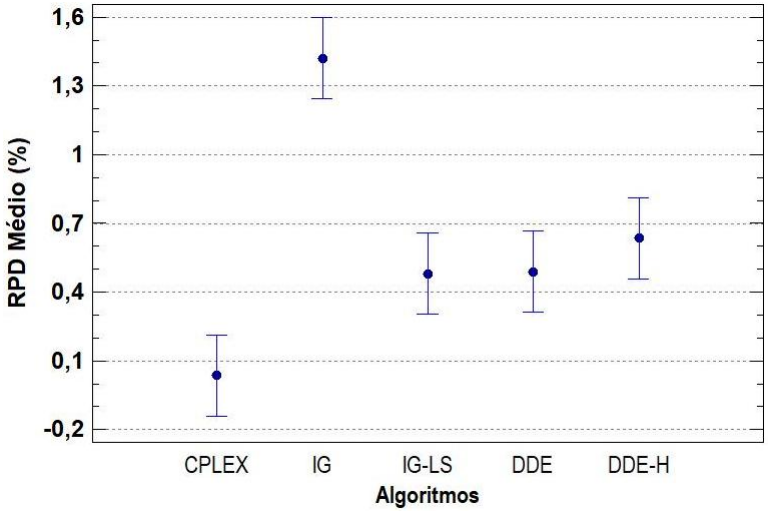


Figura 5.4: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as instâncias de tamanho $n = 20$

A Figura 5.4 apresenta, para as instâncias de pequeno porte com $n = 20$, o gráfico de médias resultante do teste HSD de Tukey com nível de confiança de 95%, o qual compara os algoritmos propostos IG, IG-LS, DDE, DDE-H com o algoritmo exato CPLEX. Observa-se que na Figura 5.4 a média do CPLEX é significativamente diferente das médias dos outros algoritmos, uma vez que o intervalo para o algoritmo CPLEX não intercepta os intervalos dos outros algoritmos. Isso nos levar a concluir que as heurísticas propostas mostram desempenho inferior ao algoritmo CPLEX.

Tabela 5.5: Desvio padrão ($n = 20$)

Instâncias	IG	IG-LS	DDE	DDE-H
J1M2F1s1	0,66	0,20	0,20	0,20
J1M2F1s2	1,24	0,41	0,31	0,40
J1M2F2s1	0,45	0,20	0,04	0,12
J1M2F2s2	0,61	0,00	0,00	0,00
J1M3F1s1	1,16	0,56	0,56	0,59
J1M3F1s2	1,88	0,29	0,32	0,64
J1M3F2s1	0,13	0,00	0,00	0,00
J1M3F2s2	0,56	0,28	0,21	0,25
J1M4F1s1	0,43	0,16	0,15	0,16
J1M4F1s2	1,24	0,20	0,46	0,67
J1M4F2s1	0,08	0,07	0,07	0,07
J1M4F2s2	1,18	0,23	0,10	0,22
J1M5F1s1	0,13	0,00	0,00	0,00
J1M5F1s2	1,17	0,41	0,32	0,52
J1M5F2s1	0,05	0,00	0,00	0,00
J1M5F2s2	1,05	0,24	0,16	0,46
Média	0,75	0,20	**0,18**	0,27

Tabela 5.6: Menores *makespan* ($n = 20$)

Instâncias	CPLEX	IG	IG-LS	DDE	DDE-H
J1M2F1s1	10	8	8	8	8
J1M2F1s2	9	6	8	7	7
J1M2F2s1	10	8	9	9	9
J1M2F2s2	10	10	10	10	10
J1M3F1s1	9	4	6	6	5
J1M3F1s2	10	5	7	7	7
J1M3F2s1	9	9	9	9	9
J1M3F2s2	8	6	9	9	9
J1M4F1s1	10	7	6	6	6
J1M4F1s2	8	3	5	5	3
J1M4F2s1	10	8	4	4	4
J1M4F2s2	7	7	8	8	8
J1M5F1s1	8	8	7	7	7
J1M5F1s2	4	4	8	7	7
J1M5F2s1	9	9	7	7	7
J1M5F2s2	7	6	8	8	8
Soma	**138**	108	119	117	114

A Tabela 5.5 mostra o desvio padrão médio de cada algoritmo para o conjunto de instância de pequeno porte com $n = 20$. O algoritmo DDE apresenta o menor desvio padrão, seguido do IG-LS, DDE-H e IG. Os algoritmos IG, IG-LS, DDE e DDE-H têm

a média global de desvio padrão para esse conjunto igual a 0,75%, 0,20%, 0,18% e 0,27%, respectivamente.

A Tabela 5.6 mostra o número de instâncias que cada algoritmo conseguiu encontrar as melhores soluções. O CPLEX encontrou soluções ótimas para 138 instâncias. Em segundo lugar, o IG-LS, que encontrou a melhor solução para 119 instâncias. O DDE, DDE-H e IG encontraram a melhor solução para 117, 114 e 108 instâncias, respectivamente.

Tabela 5.7: Resultados RPDs para as instâncias com $n = 50$ tarefas

Instância	CPLEX	IG		IG-LS		DDE		DDE-H	
n x m x F x s	F.O.	Mínimo	Médio	Mínimo	Médio	Mínimo	Médio	Mínimo	Médio
J2M2F1s1	14,33	3,27	5,11	**0,38**	1,66	1,05	1,93	0,43	**1,02**
J2M2F1s2	10,41	2,81	4,40	**0,00**	**0,96**	2,59	3,55	1,04	1,95
J2M2F2s1	10,84	1,68	3,01	**0,10**	**0,93**	0,52	1,46	0,62	1,29
J2M2F2s2	9,76	2,31	3,73	**0,00**	**0,72**	2,01	2,95	1,00	2,00
J2M3F1s1	4,01	4,29	6,42	**0,32**	1,67	0,96	1,99	0,60	**1,29**
J2M3F1s2	8,74	3,50	5,32	**0,17**	**1,50**	2,11	3,43	0,51	1,58
J2M3F2s1	4,73	1,51	3,29	**0,00**	**0,98**	0,75	1,81	0,51	1,25
J2M3F2s2	7,43	2,39	4,06	**0,30**	**1,21**	2,73	4,17	0,34	1,99
J2M4F1s1	6,53	4,74	7,32	1,14	3,01	0,89	1,79	**0,48**	**1,32**
J2M4F1s2	9,55	3,70	5,07	**0,36**	1,66	2,15	3,37	0,52	**1,64**
J2M4F2s1	9,77	2,37	4,80	**0,10**	1,48	1,29	2,59	0,96	2,00
J2M4F2s2	10,22	3,88	5,67	**0,21**	**1,41**	4,02	5,07	1,32	2,83
J2M5F1s1	8,32	6,72	8,64	0,88	3,24	1,38	2,91	**0,30**	**1,70**
J2M5F1s2	11,61	4,50	5,98	0,51	1,88	2,27	3,58	**0,37**	**1,69**
J2M5F2s1	11,61	2,88	5,02	**0,53**	**1,65**	0,75	2,67	0,62	2,39
J2M5F2s2	13,83	3,30	5,34	**0,36**	**2,02**	3,53	4,78	1,43	2,50
Média	9,48	3,37	5,20	**0,34**	**1,62**	1,81	3,00	0,69	1,78

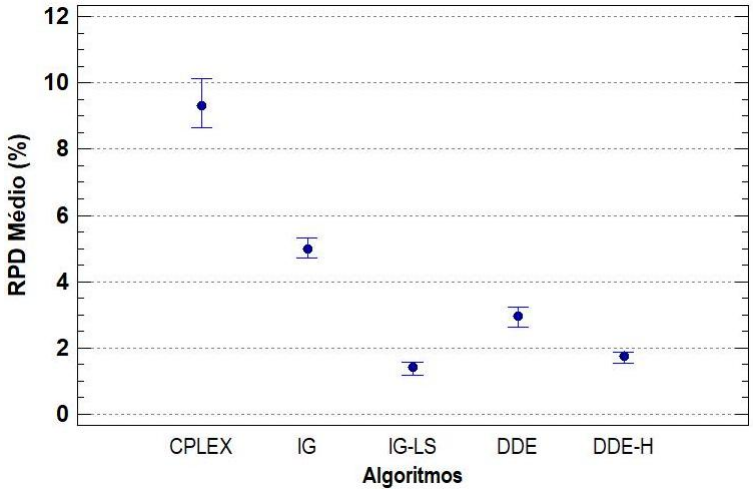


Figura 5.5: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as instâncias de tamanho $n = 50$

A Tabela 5.7 mostra os resultados para todas as instâncias de pequeno porte, com $n = 50$ tarefas. É possível observar que o CPLEX não encontrou soluções ótimas nestes conjuntos, com a maior média global de RPDs igual a 9,48%, então podemos

concluir que a partir de $n = 50$ fica inviável solucionar este problema com o modelo matemático. O algoritmo IG-LS foi o melhor, apresentando em 10 conjuntos os melhores resultados médios e uma média global de RDP igual a 1,62%. O algoritmo DDE-H também obteve um bom desempenho, sendo melhor em 6 conjuntos e com a média global de RPD igual a 1,78%. Os algoritmos DDE e IG possuem a média global de RPDs igual a 3,00% e 5,20%, respectivamente.

Na Figura 5.5 é possível observar diferenças significativas entre o CPLEX e as heurísticas propostas e a heurística da literatura. Analisando a média total de RPD, é possível verificar também que o algoritmo IG-LS obteve o menor média de RPD, acompanhado da heurística DDE-H. Nas Tabelas 5.8 e 5.9, são apresentados os desvios padrão por conjunto de instância e média global e o número de instância que cada heurística encontrou para a melhor solução, respectivamente.

Tabela 5.8: Desvio padrão ($n = 50$)

Instâncias	IG	IG-LS	DDE	DDE-H
J2M2F1s1	5,38	2,27	2,06	1,30
J2M2F1s2	8,07	3,11	5,58	3,70
J2M2F2s1	3,68	1,72	1,86	1,72
J2M2F2s2	7,83	2,36	5,14	3,87
J2M3F1s1	3,72	1,41	1,22	0,93
J2M3F1s2	5,57	2,32	3,36	2,12
J2M3F2s1	2,45	1,33	1,41	0,99
J2M3F2s2	4,60	1,84	4,06	2,93
J2M4F1s1	3,25	1,54	0,92	0,82
J2M4F1s2	3,96	1,78	2,44	1,60
J2M4F2s1	2,63	1,13	1,35	1,19
J2M4F2s2	4,64	1,77	3,45	2,59
J2M5F1s1	2,78	1,53	1,09	0,82
J2M5F1s2	3,52	1,61	2,05	1,36
J2M5F2s1	2,18	1,02	1,20	1,19
J2M5F2s2	3,50	1,77	2,68	1,75
Média	4,24	**1,78**	2,49	1,81

Tabela 5.9: Menores *makespan* ($n = 50$)

Instâncias	CPLEX	IG	IG-LS	DDE	DDE-H
J2M2F1s1	0	0	6	3	7
J2M2F1s2	0	0	10	0	0
J2M2F2s1	0	1	8	3	4
J2M2F2s2	0	0	10	0	0
J2M3F1s1	0	0	6	2	2
J2M3F1s2	0	0	8	0	5
J2M3F2s1	0	2	9	4	4
J2M3F2s2	0	1	6	0	5
J2M4F1s1	0	0	4	4	5
J2M4F1s2	0	0	7	1	6
J2M4F2s1	0	2	9	1	4
J2M4F2s2	0	0	8	0	2
J2M5F1s1	0	0	5	3	7
J2M5F1s2	0	0	5	0	6
J2M5F2s1	0	0	8	4	3
J2M5F2s2	0	0	8	0	3
Média	0	6	**117**	25	63

Nas Tabela 5.8 são apresentados os desvios padrão. O IG-LS obteve a melhor média global, com 1,78, e o DDE-H a segunda melhor média global, com 1,81. O DDE e o IG apresentaram médias de 2,48 e 4,24, respectivamente. A Tabela 5.9 mostra o número de instância que cada heurística encontrou a melhor solução. O IG-LS encontrou em 117 das 160 instâncias, o que presenta um percentual de 73,12%. O DDE-H encontrou a melhor solução para 63 instâncias, representando um percentual de 39,37%. Já os algoritmos DDE e IG encontraram para apenas 25 (15,62%) e 6 (3,75%), respectivamente. O CPLEX não encontrou soluções ótimas, nem sequer as melhores soluções para nenhuma das instâncias desse grupo.

5.4.2 Resultados Experimentais para as Instâncias de Grande Porte

Nesta seção, é feita uma comparação entre o desempenho dos algoritmos propostos neste trabalho IG, IG-LS e DDE-H e o algoritmo reimplementado DDE para as instâncias de grande porte (instâncias da literatura). O critério de parada utilizado

pelas heurísticas implementadas neste trabalho é baseado na quantidade de tempo de CPU, sendo eles: $n \cdot k$ segundos, em que n é o número de tarefas e $k = 0,75$. Para resolver cada instância, cada algoritmo foi executado 5 vezes. E para uma melhor análise dos resultados, as instâncias foram agrupadas em função dos parâmetros do problema.

Tabela 5.10: Resultados RPDs para as instâncias com $n = 100$ tarefas

Instância	IG		IG-LS		DDE		DDE-H	
n x m x F x s	Mínimo	Média	Mínimo	Média	Mínimo	Média	Mínimo	Média
J3M2F3s1	7,61	9,16	3,57	5,06	2,26	2,92	**0,00**	**0,81**
J3M2F3s2	5,80	6,86	1,79	3,18	2,84	3,55	**0,00**	**0,83**
J3M2F4s1	4,63	6,45	1,53	2,81	2,63	3,47	**0,00**	**0,98**
J3M2F4s2	4,63	5,89	0,66	2,06	3,09	4,03	**0,11**	**1,09**
J3M2F5s1	1,73	3,20	**0,42**	**1,22**	2,31	3,19	0,49	1,52
J3M2F5s2	3,39	4,50	**0,15**	**1,17**	3,27	4,42	0,51	1,56
J3M3F3s1	9,42	10,46	4,17	5,74	1,81	2,49	**0,00**	**0,75**
J3M3F3s2	6,75	7,83	3,33	4,81	3,34	3,89	**0,00**	**0,69**
J3M3F4s1	4,69	6,03	1,82	3,41	2,35	3,33	**0,00**	**0,71**
J3M3F4s2	5,19	6,51	2,18	3,35	3,53	4,51	**0,00**	**1,15**
J3M3F5s1	1,73	2,86	**0,26**	1,24	2,08	3,07	0,29	**1,21**
J3M3F5s2	2,95	4,34	0,61	1,88	3,71	4,85	**0,17**	**1,31**
J3M4F3s1	9,11	10,50	5,83	7,04	2,20	3,00	**0,00**	**0,76**
J3M4F3s2	7,11	8,16	3,80	5,01	3,19	3,79	**0,00**	**0,87**
J3M4F4s1	4,87	7,03	2,72	4,03	2,05	3,34	**0,00**	**0,95**
J3M4F4s2	5,83	6,96	3,08	4,21	3,55	4,55	**0,00**	**1,22**
J3M4F5s1	2,13	3,55	**0,35**	**1,46**	2,59	3,74	0,80	1,90
J3M4F5s2	3,33	4,83	1,17	2,31	3,79	4,64	**0,08**	**1,22**
J3M5F3s1	9,40	10,90	5,60	7,44	2,31	3,03	**0,00**	**0,92**
J3M5F3s2	6,59	7,89	3,71	4,95	3,06	3,68	**0,00**	**0,90**
J3M5F4s1	5,58	7,03	2,86	4,27	2,73	4,00	**0,00**	**1,26**
J3M5F4s2	5,36	6,73	2,54	3,61	3,11	3,94	**0,00**	**0,80**
J3M5F5s1	1,94	3,60	0,15	1,58	2,62	3,71	**0,28**	**1,43**
J3M5F5s2	3,06	4,16	0,89	2,12	3,16	4,52	**0,10**	**1,43**
Média	5,12	6,48	2,22	3,50	2,82	3,74	**0,12**	**1,09**

Neste experimento, os algoritmos foram executados sobre o conjunto com $n = 100$ tarefas. Esse conjunto ainda foi dividido em subconjuntos, que representam os fatores apresentados na Seção 5.1. Cada um desses subconjuntos contém 10 instâncias. A Tabela 5.10 apresenta os resultados obtidos nesse conjunto, sendo que a primeira coluna representa o subconjunto ao qual é aplicado o algoritmo. A coluna Mínimo significa que o valor utilizado na variável $f_{algoritmo}$ do cálculo do RPD (Equação 5.1) normalmente corresponde ao valor da média *makespan*, na verdade, para esse resultado, utilizamos o menor valor do algoritmo encontrado nas 5 execuções.

Na Tabela 5.10 é possível observar que o algoritmo DDE-H foi muito superior aos outros algoritmos, sendo melhor em 21 subconjuntos, não sendo melhor em apenas 3, em que o IG-LS foi melhor. Além disso, mostrou a melhor média global, com 1,09, seguido dos algoritmos IG-LS, DDE e IG com 3,50, 3,74 e 6,48, respectivamente.

Os resultados apresentados na Tabela 5.10 podem ser mais bem analisados na Fi-

gura 5.6, que ilusta as médias e os intervalos HSD de Tukey com nível de confiança de 95%, obtidos na execução de um teste de Análise de Variância. Pode-se notar, observando a Figura 5.6, que os resultados apresentados são significativamente diferentes e que a heurística DDE-H obteve os melhores resultados, isto é, apresentou maior melhoria em relação aos demais algoritmos.

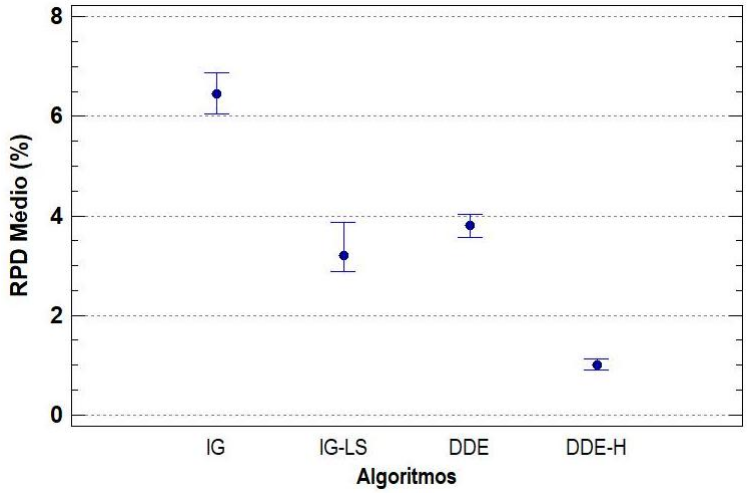


Figura 5.6: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as instâncias de tamanho $n = 100$

Tabela 5.11: Desvio padrão ($n = 100$)

Instâncias	IG	IG-LS	DDE	DDE-H
J3M2F3s1	15,93	8,51	4,73	2,61
J3M2F3s2	22,02	11,44	10,69	4,75
J3M2F4s1	12,66	5,99	6,24	3,59
J3M2F4s2	20,18	9,12	12,73	6,00
J3M2F5s1	8,60	3,64	6,82	4,30
J3M2F5s2	16,76	6,92	14,82	7,75
J3M3F3s1	10,16	5,66	2,39	1,24
J3M3F3s2	14,12	8,78	6,42	2,28
J3M3F4s1	6,64	4,21	3,56	1,50
J3M3F4s2	12,40	6,62	7,94	3,88
J3M3F5s1	4,08	2,24	3,82	2,26
J3M3F5s2	9,51	5,26	9,24	4,39
J3M4F3s1	7,78	4,88	2,15	1,11
J3M4F3s2	11,14	6,65	4,86	2,19
J3M4F4s1	6,00	3,48	2,79	1,50
J3M4F4s2	9,93	5,95	6,13	3,01
J3M4F5s1	4,02	2,07	3,49	2,48
J3M4F5s2	8,22	4,31	6,62	3,23
J3M5F3s1	6,29	4,14	1,69	0,96
J3M5F3s2	8,45	5,09	3,60	1,62
J3M5F4s1	4,55	2,68	2,54	1,44
J3M5F4s2	7,65	4,16	4,13	1,68
J3M5F5s1	3,23	1,78	2,70	1,71
J3M5F5s2	5,46	3,26	5,34	2,65
Média	9,83	5,28	5,64	**2,84**

Tabela 5.12: Menores *makespan* ($n = 100$)

Instâncias	IG	IG-LS	DDE	DDE-H
J3M2F3s1	0	0	0	10
J3M2F3s2	0	0	0	10
J3M2F4s1	0	0	0	10
J3M2F4s2	0	4	0	7
J3M2F5s1	1	6	0	4
J3M2F5s2	0	6	0	4
J3M3F3s1	0	0	2	10
J3M3F3s2	0	0	0	10
J3M3F4s1	0	0	0	10
J3M3F4s2	0	0	0	10
J3M3F5s1	1	6	0	6
J3M3F5s2	0	3	0	7
J3M4F3s1	0	0	0	10
J3M4F3s2	0	0	0	10
J3M4F4s1	0	0	0	10
J3M4F4s2	0	0	0	10
J3M4F5s1	1	6	0	4
J3M4F5s2	0	4	0	7
J3M5F3s1	0	0	0	10
J3M5F3s2	0	0	0	10
J3M5F4s1	0	0	0	10
J3M5F4s2	0	0	0	10
J3M5F5s1	0	6	0	6
J3M5F5s2	0	2	0	9
Soma	3	43	2	**204**

Na Tabela 5.11, em que é apresentado o desvio padrão por algoritmo, observa-se que a heurística DDE-H se sobressaiu em quase todas os subconjuntos, ficando atrás apenas do IG-LS, que foi melhor em apenas 2 subconjuntos. O algoritmo DDE-H

obteve a melhor média global, com 2,84, e o IG-LS, média global com 5,28. O DDE e o IG apresentaram médias 5,28 e 5,64, respectivamente.

A Tabela 5.12 mostra o número de instâncias com o qual cada heurística encontrou a melhor solução para o conjunto com $n = 100$. De um conjunto com 240 instâncias, o DDE-H encontrou os melhores resultados em 204 instâncias, o que presenta um percentual de 85% das instâncias desse conjunto. O IG-LS encontrou a melhor solução para 43 instâncias, o que representa 17,91% das instâncias. Já os algoritmos IG e DDE encontraram para apenas em 3 instâncias (1,25%) e 2 instâncias (0,83%), respectivamente.

Podemos afirmar então que, para o conjunto de $n = 100$ tarefas, o desempenho do algoritmo proposto DDE-H foi muito superior aos outros algoritmos.

O experimento acima foi repetido, e os algoritmos foram executados sobre os conjuntos com $n = 200$ e $n = 300$ tarefas. Na Tabela 5.13, observa-se claramente que o algoritmo DDE-H teve um desempenho superior aos demais algoritmos. O DDE-H conseguiu encontrar a melhor solução em todas as instâncias, o que pode ser observado na Tabela 5.15. A 5.14 mostra que seu desvio padrão foi menor, com média global de 3,26.

Tabela 5.13: Resultados RPDs para as instâncias com $n = 200$ tarefas

	IG		IG-LS		DDE		DDE-H	
Instâncias	Mínimo	Média	Mínimo	Média	Mínimo	Média	Mínimo	Média
J4M2F3s1	13,67	14,92	9,61	10,67	1,48	1,53	**0,00**	**0,26**
J4M2F3s2	10,25	10,81	7,07	7,71	3,17	3,52	**0,00**	**0,69**
J4M2F4s1	12,16	13,00	7,67	8,83	2,15	2,44	**0,00**	**0,49**
J4M2F4s2	8,40	9,25	5,93	6,79	2,85	3,12	**0,00**	**0,57**
J4M2F5s1	8,72	9,96	4,85	5,77	3,11	3,55	**0,00**	**0,78**
J4M2F5s2	7,24	8,24	4,71	5,81	3,81	4,51	**0,00**	**0,79**
J4M3F3s1	14,81	15,68	11,94	13,25	1,67	1,80	**0,00**	**0,39**
J4M3F3s2	9,33	10,11	8,25	9,20	2,34	2,80	**0,00**	**0,49**
J4M3F4s1	12,60	13,41	9,52	10,82	2,10	2,51	**0,00**	**0,43**
J4M3F4s2	8,06	8,91	7,25	8,18	2,99	3,44	**0,00**	**0,68**
J4M3F5s1	8,94	9,81	6,08	7,27	2,94	3,61	**0,00**	**0,80**
J4M3F5s2	7,10	7,83	5,79	6,74	3,29	3,91	**0,00**	**0,59**
J4M4F3s1	14,35	15,48	12,79	13,81	1,74	1,88	**0,00**	**0,45**
J4M4F3s2	9,87	10,43	8,17	9,40	2,69	3,00	**0,00**	**0,61**
J4M4F4s1	12,09	12,95	9,96	11,10	2,47	2,87	**0,00**	**0,49**
J4M4F4s2	7,92	8,81	7,28	8,20	3,27	3,77	**0,00**	**0,63**
J4M4F5s1	8,84	10,06	6,48	7,43	3,20	3,93	**0,00**	**0,75**
J4M4F5s2	7,09	8,01	5,77	6,80	3,84	4,30	**0,00**	**0,88**
J4M5F3s1	14,61	15,64	12,28	13,60	1,75	1,82	**0,00**	**0,35**
J4M5F3s2	9,49	10,29	8,70	9,51	2,25	2,36	**0,00**	**0,51**
J4M5F4s1	11,91	13,10	10,23	11,25	2,32	2,66	**0,00**	**0,47**
J4M5F4s2	8,42	9,12	7,40	8,37	3,18	3,54	**0,00**	**0,53**
J4M5F5s1	8,74	9,86	6,17	7,41	3,44	4,01	**0,00**	**0,73**
J4M5F5s2	7,17	7,92	5,54	6,78	3,41	3,97	**0,00**	**0,72**
Média	10,07	10,98	7,89	8,95	2,73	3,12	**0,00**	**0,59**

A Figura 5.7 ilustra melhor as médias e os intervalos de HSD de Tukey. Nela

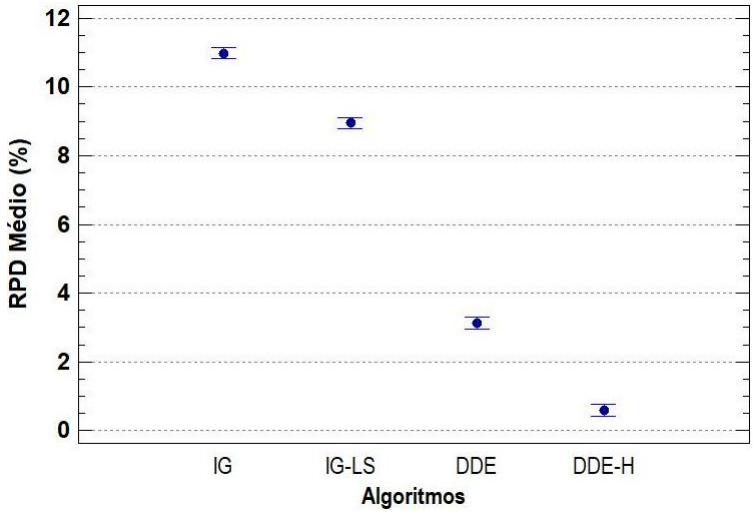


Figura 5.7: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as instâncias de tamanho $n = 200$

pode-se observar quão superior e estaticamente melhor foi algoritmo DDE-H sobre os outros algoritmos. Em resumo, os algoritmos DDE e DDE-H obtiveram uma média global de RPD igual a $3,12$ e um desvio padrão de $8,67$, não encontrando nenhuma melhor solução. O algoritmo IG-LS obteve média global de RPD e desvio padrão igual a $8,95$ e $21,78$, respectivamente. E, por fim, o algoritmo IG teve um RPD médio global de $10,07$ e um desvio padrão de $29,80$.

Tabela 5.14: Desvio padrão ($n = 200$)

Instâncias	IG	IG-LS	DDE	DDE-H
J4M2F3s1	48,84	31,91	4,54	1,69
J4M2F3s2	66,04	43,53	19,82	7,51
J4M2F4s1	44,27	27,84	7,69	3,30
J4M2F4s2	58,99	39,80	18,18	7,02
J4M2F5s1	36,84	19,63	12,10	4,81
J4M2F5s2	55,06	36,64	27,81	9,69
J4M3F3s1	28,84	22,46	3,05	1,30
J4M3F3s2	34,74	29,22	9,07	3,01
J4M3F4s1	25,97	19,41	4,58	1,76
J4M3F4s2	31,59	26,64	11,39	4,48
J4M3F5s1	20,69	14,36	7,20	3,06
J4M3F5s2	29,34	23,21	13,53	4,81
J4M4F3s1	21,75	17,76	2,43	1,19
J4M4F3s2	27,39	22,98	7,31	2,93
J4M4F4s1	19,12	15,12	3,95	1,42
J4M4F4s2	24,08	20,52	9,53	3,06
J4M4F5s1	16,18	11,02	5,96	2,21
J4M4F5s2	23,04	18,12	11,21	4,73
J4M5F3s1	17,13	13,69	1,83	0,72
J4M5F3s2	21,05	17,78	4,43	1,84
J4M5F4s1	15,12	11,87	2,84	1,10
J4M5F4s2	19,26	16,33	6,90	2,26
J4M5F5s1	12,33	8,69	4,67	1,64
J4M5F5s2	17,63	14,16	8,13	2,70
Média	29,80	21,78	8,67	**3,26**

Tabela 5.15: Menores *makespan* ($n = 200$)

Instâncias	IG	IG-LS	DDE	DDE-H
J4M2F3s1	0	0	0	10
J4M2F3s2	0	0	0	10
J4M2F4s1	0	0	0	10
J4M2F4s2	0	0	0	10
J4M2F5s1	0	0	0	10
J4M2F5s2	0	0	0	10
J4M3F3s1	0	0	0	10
J4M3F3s2	0	0	0	10
J4M3F4s1	0	0	0	10
J4M3F4s2	0	0	0	10
J4M3F5s1	0	0	0	10
J4M3F5s2	0	0	0	10
J4M4F3s1	0	0	0	10
J4M4F3s2	0	0	0	10
J4M4F4s1	0	0	0	10
J4M4F4s2	0	0	0	10
J4M4F5s1	0	0	0	10
J4M4F5s2	0	0	0	10
J4M5F3s1	0	0	0	10
J4M5F3s2	0	0	0	10
J4M5F4s1	0	0	0	10
J4M5F4s2	0	0	0	10
J4M5F5s1	0	0	0	10
J4M5F5s2	0	0	0	10
Soma	0	0	0	**240**

Para o conjunto com $n = 300$, os experimentos têm comportamento semelhante ao conjunto $n = 200$. Na Tabela 5.16, também é possível observar que o algoritmo DDE-

H teve o melhor desempenho entre os demais algoritmos, com média RPD global de 0,39. O DDE-H também conseguiu encontrar a melhor solução em todas as instâncias desse conjunto, o que é mostrado na Tabela 5.18. A Tabela 5.17 mostra que seu desvio padrão foi menor, com média global de 3,10.

Tabela 5.16: Resultados RPDs para as instâncias com $n = 300$ tarefas

	IG		IG-LS		DDE		DDE-H	
Instâncias	Mínimo	Média	Mínimo	Média	Mínimo	Média	Mínimo	Média
J5M2F3s1	16,47	17,09	13,77	15,06	1,17	1,17	**0,00**	**0,22**
J5M2F3s2	11,33	12,06	9,94	10,92	2,32	2,34	**0,00**	**0,41**
J5M2F4s1	15,12	15,74	12,94	13,84	1,75	1,75	**0,00**	**0,38**
J5M2F4s2	10,27	10,86	9,42	10,11	2,73	2,83	**0,00**	**0,42**
J5M2F5s1	11,92	12,58	9,89	10,75	2,37	2,57	**0,00**	**0,40**
J5M2F5s2	8,74	9,46	7,64	8,43	2,66	2,98	**0,00**	**0,50**
J5M3F3s1	16,96	17,73	15,65	16,51	1,18	1,18	**0,00**	**0,27**
J5M3F3s2	10,73	11,51	10,22	10,89	2,03	2,04	**0,00**	**0,33**
J5M3F4s1	15,23	16,13	14,34	15,04	2,07	2,07	**0,00**	**0,32**
J5M3F4s2	9,84	10,41	9,28	9,88	2,66	2,76	**0,00**	**0,48**
J5M3F5s1	12,24	13,26	11,06	12,06	2,67	2,80	**0,00**	**0,58**
J5M3F5s2	8,24	8,88	7,86	8,54	3,05	3,18	**0,00**	**0,44**
J5M4F3s1	16,41	17,28	15,32	16,43	1,17	1,17	**0,00**	**0,22**
J5M4F3s2	11,22	11,75	10,79	11,40	2,06	2,07	**0,00**	**0,30**
J5M4F4s1	15,38	16,09	14,48	15,28	2,14	2,14	**0,00**	**0,33**
J5M4F4s2	9,84	10,46	9,61	10,26	2,29	2,36	**0,00**	**0,38**
J5M4F5s1	12,33	13,06	10,84	11,74	2,63	2,72	**0,00**	**0,54**
J5M4F5s2	8,46	9,05	7,88	8,51	3,29	3,54	**0,00**	**0,54**
J5M5F3s1	16,62	17,37	15,64	16,37	1,27	1,27	**0,00**	**0,24**
J5M5F3s2	10,89	11,31	10,29	11,16	2,49	2,57	**0,00**	**0,35**
J5M5F4s1	15,14	15,91	14,11	14,66	1,98	2,00	**0,00**	**0,37**
J5M5F4s2	9,93	10,48	9,30	9,67	2,66	2,78	**0,00**	**0,40**
J5M5F5s1	12,36	13,31	10,85	11,38	2,57	2,72	**0,00**	**0,49**
J5M5F5s2	8,21	8,82	7,08	7,82	2,99	3,09	**0,00**	**0,47**
Média	12,24	12,94	11,17	11,95	2,26	2,34	**0,00**	**0,39**

A Figura 5.8 apresenta de forma mais clara a Tabela 5.16 com as médias e os intervalos de HSD de Tukey. Nela pode-se observar quão superior e estaticamente melhor foi algoritmo DDE-H sobre os outros algoritmos. Em resumo, os algoritmos DDE e DDE-H obtiveram uma média global de RPD igual a 2,34 e um desvio padrão de 9,29, não encontrando nenhuma melhor solução. O algoritmo IG-LS obteve média global de RPD e desvios padrão igual a 11,95 e 42,38, respectivamente. E, por fim, o algoritmo IG teve um RPD médio global de 12,24 e um desvio padrão de 50,31.

5.4.3 Análise de Convergência dos Algoritmos heurísticos

Os experimentos descritos nas Seções 5.4.1 e 5.4.2 foram realizados focando na qualidade das soluções em relação ao valor da função objetivo. No entanto, é importante verificar a qualidade das heurísticas em relação ao tempo de execução. Por essa razão, duas instâncias com diferentes níveis de fatores foram selecionadas, I_1 e I_2. As

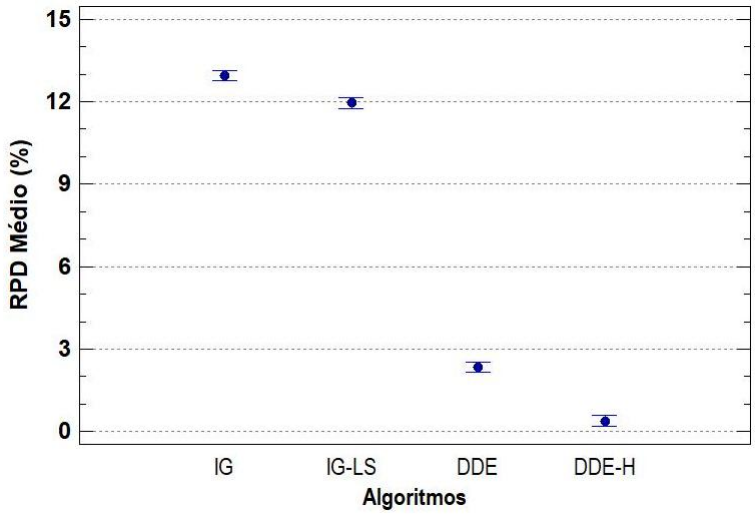


Figura 5.8: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para as instâncias de tamanho $n = 300$

Tabela 5.17: Desvio padrão ($n = 300$)

Instâncias	IG	IG-LS	DDE	DDE-H
J5M2F3s1	79,93	64,86	4,98	2,17
J5M2F3s2	109,60	91,25	19,28	6,15
J5M2F4s1	76,10	61,43	7,76	3,40
J5M2F4s2	101,27	86,28	24,08	7,65
J5M2F5s1	64,83	50,82	12,21	4,02
J5M2F5s2	91,83	75,11	26,73	9,06
J5M3F3s1	46,83	39,87	2,82	1,26
J5M3F3s2	58,59	50,64	9,39	2,86
J5M3F4s1	44,24	37,65	5,16	1,76
J5M3F4s2	54,26	47,01	13,19	4,59
J5M3F5s1	38,80	32,27	7,51	3,24
J5M3F5s2	48,01	42,26	15,67	4,57
J5M4F3s1	34,92	30,35	2,14	0,81
J5M4F3s2	45,57	40,37	7,33	2,26
J5M4F4s1	33,49	29,10	4,06	1,32
J5M4F4s2	41,71	37,46	8,62	2,78
J5M4F5s1	28,90	23,89	5,51	2,22
J5M4F5s2	37,46	32,14	13,43	4,13
J5M5F3s1	27,32	23,52	1,82	0,78
J5M5F3s2	34,10	30,86	7,08	2,09
J5M5F4s1	25,86	21,75	2,96	1,03
J5M5F4s2	32,42	27,28	7,88	2,27
J5M5F5s1	23,04	17,93	4,30	1,41
J5M5F5s2	28,42	23,03	9,00	2,66
Média	50,31	42,38	9,29	**3,10**

Tabela 5.18: Menores *makespan* ($n = 300$)

Instâncias	IG	IG-LS	DDE	DDE-H
J5M2F3s1	0	0	0	10
J5M2F3s2	0	0	0	10
J5M2F4s1	0	0	0	10
J5M2F4s2	0	0	0	10
J5M2F5s1	0	0	0	10
J5M2F5s2	0	0	0	10
J5M3F3s1	0	0	0	10
J5M3F3s2	0	0	0	10
J5M3F4s1	0	0	0	10
J5M3F4s2	0	0	0	10
J5M3F5s1	0	0	0	10
J5M3F5s2	0	0	0	10
J5M4F3s1	0	0	0	10
J5M4F3s2	0	0	0	10
J5M4F4s1	0	0	0	10
J5M4F4s2	0	0	0	10
J5M4F5s1	0	0	0	10
J5M4F5s2	0	0	0	10
J5M5F3s1	0	0	0	10
J5M5F3s2	0	0	0	10
J5M5F4s1	0	0	0	10
J5M5F4s2	0	0	0	10
J5M5F5s1	0	0	0	10
J5M5F5s2	0	0	0	10
Soma	0	0	0	**240**

principais características das instâncias utilizadas, número de tarefas (n), máquinas (m), número de famílias (F) e tamanho das tarefas (s_j), são descritas abaixo.

- I_1: $n = 100$, $m = 3$, $F = 6$ e $s = [10, 30]$
- I_2: $n = 200$, $m = 4$, $F = 12$ e $s = [10, 30]$

O experimento consiste em fixar um valor de tempo e executar os algoritmos a partir da mesma solução inicial. Assim, em cada melhoria obtida por uma heurística, o resultado e o tempo despendido são armazenados para realizar a análise da convergência das heurísticas em função do tempo. Ao final, foi gerado o gráfico tempo

computacional (segundos) versus *makespan* para cada instância. Os resultados são apresentados nas Figuras 5.9 e 5.10.

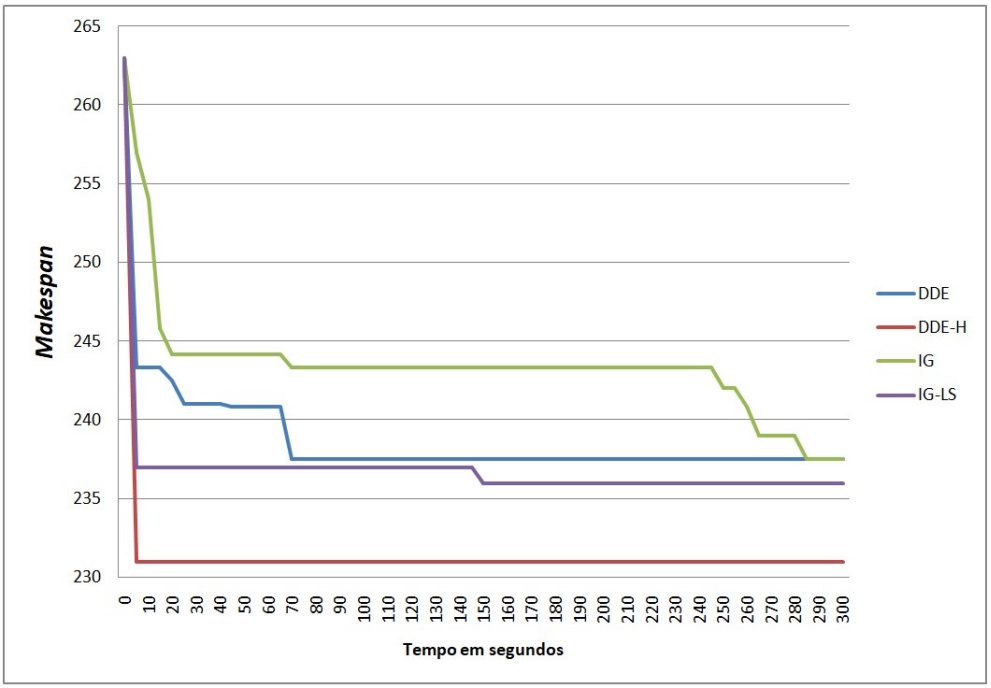


Figura 5.9: Convergência das soluções da instância I_1 ao decorrer do tempo

Ao analisar o gráfico da Figura 5.9, é fácil observar que os algoritmos DDE encontram a melhor solução mais rapidamente que os outros algoritmos propostos. O DDE-H obtém a melhor solução em menos de 60 segundos, enquanto os algoritmos IGs despendem mais tempo em suas buscas. Nota-se, também, que, embora o algoritmo IG-LS apresente uma convergência maior nos primeiro instantes de execução, ele demorou mais tempo para encontrar a melhor solução do que o algoritmo DDE-H. Sendo assim, neste caso, entende-se que o DDE-H tem a possibilidade de encontrar a melhor solução mais rapidamente.

Como pode ser observado na Figura 5.10, o DDE consegue encontrar soluções boas com o menor tempo de execução se comparado a outros algoritmos. Mas, neste caso, o DDE-H foi o que mais demorou a encontrar a melhor solução com aproximadamente 300 segundos. O DDE novamente foi o primeiro a encontrar a melhor solução, com menos de 30 segundos.

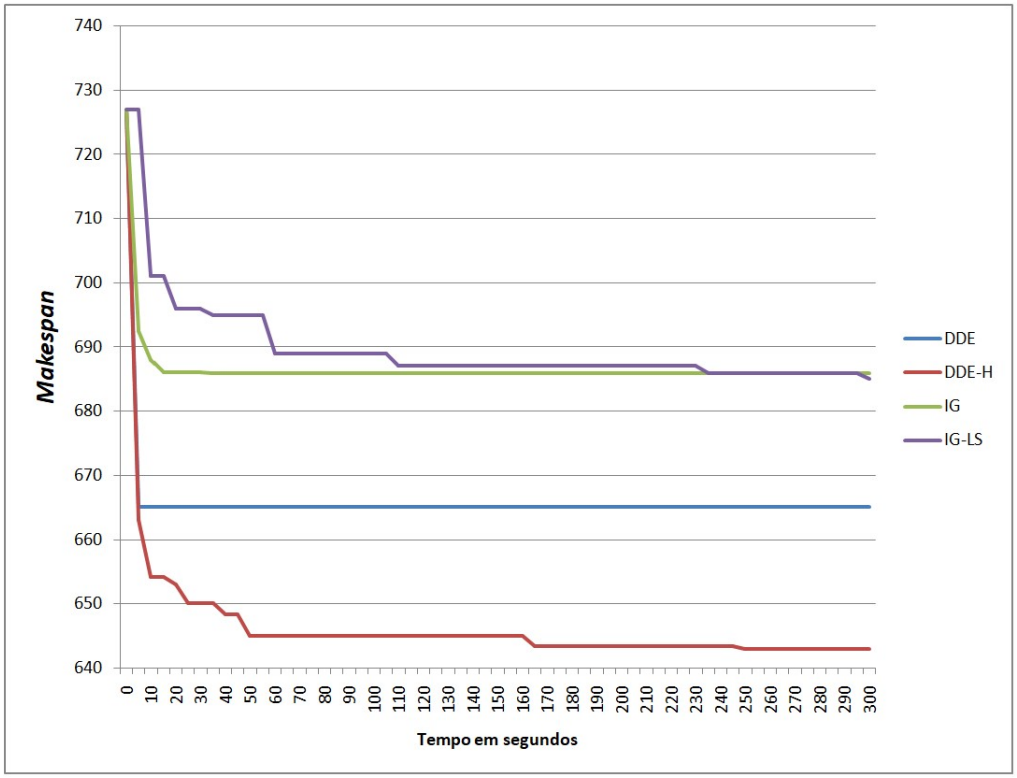


Figura 5.10: Convergência das soluções da instância I_2 ao decorrer do tempo

Capítulo 6

Conclusão

Nesse trabalho foi abordado o problema de sequenciamento em máquinas paralelas uniformes de processamento em lote. O problema considera a minimização do máximo tempo de conclusão do processamento das tarefas, o *makespan*, levando em conta que as tarefas podem ser processadas simultaneamente dentro de lotes com tarefas de mesma família. Este é um importante problema, pois sua aplicação prática pode ser vista na indústria.

Uma adaptação da formulação matemática do problema apresentada por Zhou (Zhou et al., 2016), foi proposta para o problema $Q_m \mid p_j, s_j, b, v_i, Q_i, incompatible \mid C_{max}$. Utilizando o software IBM CPLEX, resolveu-se o modelo matemático para 320 instâncias de pequeno porte. Para as outras 720 instâncias restantes, foi aplicada uma abordagem heurística. Esta última abordagem não garante que a solução encontrada seja ótima. Os resultados para as 720 instâncias foram satisfatórios, uma vez que a abordagem encontrou soluções de qualidade.

O problema é classificado como NP-difícil, o que justifica o estudo de algoritmos heurísticos que obtenham soluções em tempo computacional razoável. Foram propostos três algoritmos heurísticos. Os dois primeiros são baseados na meta-heurística *Iterated Greedy* (IG). O primeiro IG é aplicado a uma iteração destrutivo-construtiva, e um segundo IG-LS é aplicado uma busca local. O terceiro é um método híbrido baseado na meta-heurística *Discrete Differential Evolution* (DDE), chamado de DDE-Híbrido, com um conjunto elite e uma fase de busca local que é aplicada à meta-heurística IG.

O algoritmo *Discrete Differential Evolution* (DDE) proposto por Zhou (Zhou et al., 2016) para um problema próximo ($Q_m \mid p_j, s_j, b, v_i, Q_i, \mid C_{max}$), foi reimplementado e usado em comparação. Vale ressaltar que o DDE apresenta desempenhos muito bons, em comparação com alguns algoritmos propostos. Os resultados obtidos foram validados por testes estatísticos.

Os resultados obtidos indicam claramente que os algoritmos propostos são altamente eficientes quando comparados com o modelo matemático e com o algoritmo

DDE. O algoritmo IG-LS foi superior aos demais algoritmo para os conjuntos de pequeno porte. E para um conjunto de grande porte, com $n = 100$, foi obtido resultados superiores ao DDE. O algoritmo com o melhor desempenho no conjunto de grande porte, foi o algoritmo proposto DDE-H, tendo um desempenho muito superior que dos demais algoritmos, nos conjuntos com $n = 200$ e $n = 300$ tarefas.

Pode-se concluir que os algoritmos heurísticos propostos neste trabalho são validados para resolver este tipo de problema de sequenciamento, como visto pelos resultados obtidos com a melhor meta-heurística disponível na literatura em um amplo conjunto de instâncias. Da mesma forma, acredita-se que os resultados apresentados sejam os melhores até o momento na literatura para o problema em estudo.

6.1 Trabalhos Futuros

A primeira etapa da pesquisa gerou um artigo publicado no L SBPO (Tavares and Arroyo, 2018), em que foi estudado um problema mais simples, com algumas abordagens iniciais desenvolvidas durante a pesquisa. O problema apresentado nesse artigo era de BPMs paralelas uniformes, com capacidades diferentes e tarefas de diferentes tamanhos, não tendo sido consideradas as famílias de tarefas. Para solucionar foram apresentados uma primeira versão do modelo matemático e dois algoritmos heurísticos baseados em IG e DDE.

Pretende-se, como trabalhos futuros, acrescentar na modelagem novas restrições tornando o problema cada vez mais próximo da realidade. Essas novas restrições incluem *setup* de preparação das máquinas entre lotes, tempo de processamento de tarefas diferente para cada máquina e tempo de chegada dinâmico de cada tarefa ou famílias de tarefas. Por fim, restrições sobre o tempo de entrega de cada tarefa.

Referências Bibliográficas

Arroyo, J. E. C. and Leung, J. Y.-T. (2017a). An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. *Computers & Industrial Engineering*, 105:84–100.

Arroyo, J. E. C. and Leung, J. Y.-T. (2017b). Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. *Computers & Operations Research*, 78:117–128.

Chang, P.-Y., Damodaran*, P., and Melouk, S. (2004). Minimizing makespan on parallel batch processing machines. *International Journal of Production Research*, 42(19):4211–4220.

Chen, H., Du, B., and Huang, G. Q. (2010). Metaheuristics to minimise makespan on parallel batch processing machines with dynamic job arrivals. *International Journal of Computer Integrated Manufacturing*, 23(10):942–956.

Cheng, B., Cai, J., Yang, S., and Hu, X. (2014). Algorithms for scheduling incompatible job families on single batching machine with limited capacity. *Computers & Industrial Engineering*, 75:116–120.

Cheng, T. E. and Kovalyov, M. Y. (1995). Single machine batch scheduling with deadlines and resource dependent processing times. *Operations Research Letters*, 17(5):243–249.

Chiang, T.-C., Cheng, H.-C., and Fu, L.-C. (2010). A memetic algorithm for minimizing total weighted tardiness on parallel batch machines with incompatible job families and dynamic job arrival. *Computers & Operations Research*, 37(12):2257–2269.

Chung, S., Tai, Y., and Pearn, W. (2009). Minimising makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes. *International Journal of Production Research*, 47(18):5109–5128.

Damodaran, P. and Chang, P.-Y. (2008). Heuristics to minimize makespan of parallel batch processing machines. *The International Journal of Advanced Manufacturing Technology*, 37(9):1005–1013.

Damodaran, P., Diyadawagamage, D. A., Ghrayeb, O., and Vélez-Gallego, M. C. (2012). A particle swarm optimization algorithm for minimizing makespan of nonidentical parallel batch processing machines. *The International Journal of Advanced Manufacturing Technology*, 58(9-12):1131–1140.

Damodaran, P. and Vélez-Gallego, M. C. (2012). A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert Systems with Applications*, 39(1):1451–1458.

Damodaran, P., Vélez-Gallego, M. C., and Maya, J. (2011). A grasp approach for makespan minimization on parallel batch processing machines. *Journal of Intelligent Manufacturing*, 22(5):767–777.

Dobson, G. and Nambimadom, R. S. (2001). The batch loading and scheduling problem. *Operations research*, 49(1):52–65.

Fu, R., Tian, J., and Yuan, J. (2009). On-line scheduling on an unbounded parallel batch machine to minimize makespan of two families of jobs. *Journal of Scheduling*, 12(1):91.

Jacoba, V. V. and Arroyo, J. E. C. (2016). Ils heuristics for the single machine scheduling problem with sequence dependent family setup times to minimize total tardiness. *??*, 1(1):1–21.

Jia, Z.-h., Wang, C., and Leung, J. Y.-T. (2016). An aco algorithm for makespan minimization in parallel batch machines with non-identical job sizes and incompatible job families. *Applied Soft Computing*, 38:395–404.

Kempf, K. G., Uzsoy, R., and Cheng-Shou, W. (1998). Scheduling a single batch processing machine with secondary resource constraints. *Journal of Manufacturing Systems*, 17(1):37.

Lee, C.-Y., Uzsoy, R., and Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775.

Li, X., Chen, H., Du, B., and Tan, Q. (2013a). Heuristics to schedule uniform parallel batch processing machines with dynamic job arrivals. *International Journal of Computer Integrated Manufacturing*, 26(5):474–486.

Li, X., Huang, Y., Tan, Q., and Chen, H. (2013b). Scheduling unrelated parallel batch processing machines with non-identical job sizes. *Computers & Operations Research*, 40(12):2983–2990.

Liu, L., Ng, C., and Cheng, T. E. (2010). On the complexity of bi-criteria scheduling on a single batch processing machine. *Journal of scheduling*, 13(6):629–638.

Malve, S. and Uzsoy, R. (2007a). A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Computers & Operations Research*, 34(10):3016–3028.

Malve, S. and Uzsoy, R. (2007b). A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Computers Operations Research*, 34(10):3016 – 3028.

Mathirajan, M. and Sivakumar, A. I. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9-10):990–1001.

Mehta, S. V. and Uzsoy, R. (1998). Minimizing total tardiness on a batch processing machine with incompatible job families. *IIE transactions*, 30(2):165–178.

Mönch, L., Balasubramanian, H., Fowler, J. W., and Pfund, M. E. (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers Operations Research*, 32(11):2731 – 2750.

Oulamara, A., Finke, G., and Kuiteing, A. K. (2009). Flowshop scheduling problem with a batching machine and task compatibilities. *Computers & Operations Research*, 36(2):391–401.

Pinedo, M. and Hadavi, K. (1992). Scheduling: theory, algorithms and systems development. In *Operations Research Proceedings 1991*, pages 35–42. Springer.

Potts, C. N. and Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228–249.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Tang, L., Gong, H., Liu, J., and Li, F. (2014). Bicriteria scheduling on a single batching machine with job transportation and deterioration considerations. *Naval Research Logistics (NRL)*, 61(4):269–285.

Tang, L. and Liu, P. (2009). Minimizing makespan in a two-machine flowshop scheduling with batching and release time. *Mathematical and Computer Modelling*, 49(5):1071–1077.

Tavares, R. G. and Arroyo, J. E. C. (2018). Um algoritmo iterated greedy para o sequenciamento de tarefas em máquinas paralelas uniformes de processamento em lote. *L Simpósio Brasileiro de Pesquisa Operacional; Rio de Janeiro, Brasil 2018*, 50(1):1–12.

Tian, J., Cheng, T. E., Ng, C., and Yuan, J. (2011). Online scheduling on unbounded parallel-batch machines with incompatible job families. *Theoretical Computer Science*, 412(22):2380–2386.

Uzsoy, R. (1994). Scheduling a single batch processing machine with non-identical job sizes. *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, 32(7):1615–1635.

Uzsoy, R. (1995). Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, 33(10):2685–2708.

Vallada, E., Ruiz, R., and Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4):1350–1373.

Wang, H.-M. and Chou, F.-D. (2010). Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics. *Expert Systems with Applications*, 37(2):1510–1521.

Xu, S. and Bean, J. C. (2007). A genetic algorithm for scheduling parallel non-identical batch processing machines. In *Computational Intelligence in Scheduling, 2007. SCIS'07. IEEE Symposium on*, pages 143–150. IEEE.

Yuan, J., Shang, W., and Feng, Q. (2005). A note on the scheduling with two families of jobs. *Journal of Scheduling*, 8(6):537–542.

Zhou, S., Liu, M., Chen, H., and Li, X. (2016). An effective discrete differential evolution algorithm for scheduling uniform parallel batch processing machines with non-identical capacities and arbitrary job sizes. *International Journal of Production Economics*, 179:1–11.