Um algoritmo ILS–RVND para o problema de escalonamento do tipo *flow shop* de permutação com tempo de conclusão total como medida de desempenho

Altair Jussadir da Silva Pinto

altair@eng.ci.ufpb.br

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2020

Altair Jussadir da Silva Pinto

altair@eng.ci.ufpb.br

Um algoritmo ILS–RVND para o problema de escalonamento do tipo *flow shop* de permutação com tempo de conclusão total como medida de desempenho

Dissertação apresentada ao Programa de Pós-Graduação em Informática
do Centro de Informática, da Universidade Federal da Paraíba,
como requisito parcial para a obtenção do grau de Mestre em Informática

Orientador: Prof. Dr. Lucídio dos Anjos Formiga Cabral

Julho de 2020

Catalogação na publicação
Seção de Catalogação e Classificação

P659a Pinto, Altair Jussadir da Silva.
Um algoritmo ILS-RVND para o problema de escalonamento do tipo flow shop de permutação com tempo de conclusão total como medida de desempenho / Altair Jussadir da Silva Pinto. - João Pessoa, 2020.
60 f. : il.

Orientação: Lucídio dos Anjos Formiga Cabral.
Dissertação (Mestrado) - UFPB/CI.

1. Informática. 2. Otimização combinatória. 3. Escalonamento tipo flow shop. 4. Programação linear inteira. 5. Meta-heurística. I. Cabral, Lucídio dos Anjos Formiga. II. Título.

UFPB/BC CDU 004(043)

Elaborado por Larissa Silva Oliveira de Mesquita - CRB-15/746

DEDICATÓRIA

Dedico aos meus pais, demais familiares e amigos.

Ao meu orientador, e amigo Lucídio Cabral pelos ensinamentos, paciência e disponibilidade constante desde o final da graduação e também durante o mestrado.

Aos amigos Emerson Ferreira, pelos auxílios e sugestões, e Gabriel Basso, pelas sugestões.

Aos professores Gilberto e Quirino pela avaliação e sugestões para o trabalho.

À todos que de alguma maneira contribuíram para a realização deste trabalho, meu muito obrigado.

RESUMO

Este trabalho apresenta FILS, uma nova heurística do tipo ILS–RVND para o problema de escalonamento do tipo *flow shop* com tempo de conclusão total como medida de desempenho, que produziu 55 novos melhores resultados de mínimo para um conjunto de 90 instâncias de referência apresentadas para o problema em questão. Nesse estudo, quatro diferentes parametrizações da heurística foram examinadas através de experimentos computacionais, utilizamos as instâncias de Taillard (1993), demonstrando que a abordagem proposta é robusta e efetiva. FILS foi comparado com algoritmos tais como VNS, hDDE, DABC, HGLS, VNS4, AGA e V4AGA, e os resultados indicam que o FILS é superior na maior parte dos casos.

Palavras-chave: Meta-heurísticas, Otimização Combinatória, Escalonamento do Tipo *Flow Shop*, Tempo de Conclusão Total.

ABSTRACT

This work presents FILS, a new ILS–RVND heuristic for the flow shop scheduling problem with total completion time as optimality criterion, which produced 55 best minimum values for a set of 90 reference instances presented for the problem in question. In this study, four different parameterizations for using the approach are examined through computational experiments, using the benchmark problems from Taillard (1993), demonstrating that the proposed approach is a robust and effective solution. FILS was compared to algorithms such as VNS, hDDE, DABC, HGLS, VNS4, AGA and V4AGA, where the results indicate that FILS is superior in most cases.

Key-words: Meta-heuristics, Combinatorial Optimization, Flow Shop Scheduling, Total Completion Time.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

Sumário

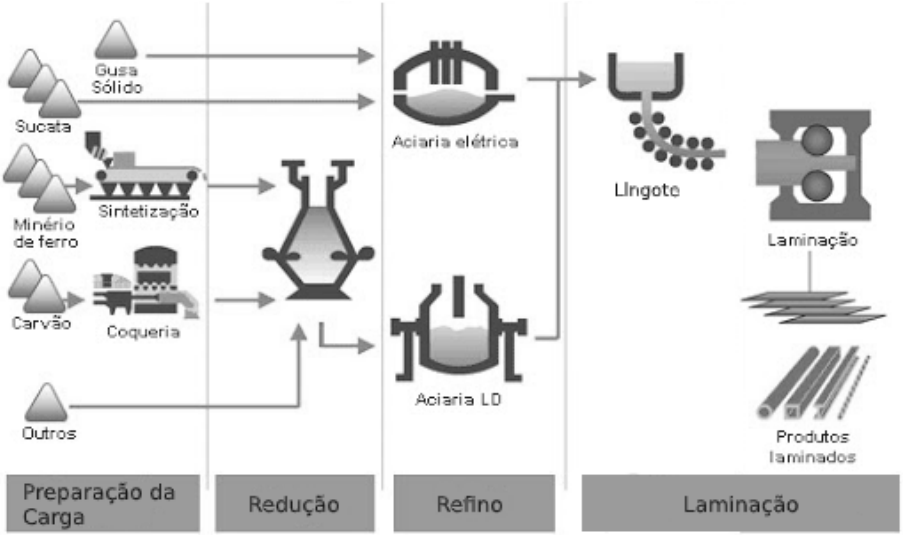
1 INTRODUÇÃO

1.1 Motivação

O problema de escalonamento do tipo *flow shop* (FSSP, *Flow Shop Scheduling Problem*) é um dos problemas de escalonamento mais conhecidos e estudados na literatura de Otimização Combinatória, começando com as pesquisas sobre escalonamentos ótimos em duas e três máquinas, feitas por Johnson [34][1]. No FSSP, temos que, em qualquer dado momento, cada máquina processa, no máximo, uma tarefa, e cada tarefa é processada por, no máximo, uma máquina; além disso, não há preempção, i.e., o processamento de uma tarefa não pode ser interrompido. O objetivo é determinar o escalonamento que otimiza uma determinada medida de desempenho.

Encontramos instâncias do FSSP em áreas estratégicas da sociedade, principalmente no setor secundário. Um exemplo é o processo de fabricação do aço, como mostra a Figura 1.

Figura 1: Processo de fabricação de aço.



Fonte: Autor

Primeiro, há a *preparação da carga*, onde minérios de ferro são aglomerados com cal e carvão — este último na forma de coque —, resultando numa mistura chamada de sinter. Após isso, passamos para a *redução*, onde essa mistura é carregada num alto-forno, onde oxigênio aquecido a cerca de 1000 $\circ$C é soprado por baixo do forno; o coque, em contato com o ar quente, libera calor suficiente para fundir a carga metálica, reduzindo o minério de ferro a um metal líquido chamado ferro-gusa. Na etapa de *refino*, aciarias

[1]Apesar de Johnson ser o primeiro, na literatura, a trabalhar no FSSP, e de Heller [30] fazer uma vaga associação entre uma classe de problemas de escalonamento e o modo como flow shops trabalham, é Heller [31] quem utiliza o termo "problema de escalonamento do tipo *flow shop*" pela primeira vez.

transformam o ferro-gusa em aço líquido; nesse processo, parte do carbono contido no ferro-gusa é removido, juntamente com impurezas. Por fim, há o processo de *laminação*, onde o aço, já solidificado na forma de lingotes, semi-acabados e blocos, são trabalhados por laminadores, que transforma-o em outros produtos siderúrgicos, e.g., correntes, fios e tubulações.

Um caso particular de FSSP é o problema de escalonamento do tipo *flow shop* de permutação (PFSSP, Permutation Flow Shop Scheduling Problem). Nessa classe, as soluções possíveis são escalonamentos de permutação, i.e., escalonamentos onde a ordem de processamento é idêntica em todas as máquinas [69]. Partindo de [31], grande parte das pesquisas feitas com o FSSP focam em encontrar soluções para o PFSSP [72].

Na literatura, temos registros de tentativas de solucionar o PFSSP de forma ótima, e.g., através de métodos de *branch and bound* [33, 11] e *programação linear inteira mista* [79]. Porém, exceto por uma classe bastante restrita de problemas [85], encontrar um escalonamento ótimo é um problema $\mathcal{NP}$-difícil, i.e., até o presente momento, não sabemos se é possível resolver o caso geral de forma ótima em tempo polinomial [43]. O PFSSP, em relação ao FSSP, reduz o tamanho do espaço de busca de $(n!)^m$ para $n!$, o que simplifica a busca; porém, ainda assim, não há redução na dificuldade do problema. Sendo assim, nossa atenção volta-se para os métodos heurísticos, algoritmos que entregam soluções de alta qualidade em um tempo computacional razoável [61].

Neste trabalho, objetivamos resolver o PFSSP tendo, como medida de desempenho, a soma dos tempos de conclusão das tarefas. Para isso, propomos uma abordagem híbrida baseada no método de busca local iterada (ILS, *Iterated Local Search*) e no método de descida aleatória em vizinhança variável (RVND, *Random Variable Neighborhood Descent*). O ILS é uma meta-heurística simples, mas eficiente; porém, seu desempenho na geração de soluções é altamente dependente do método de busca local utilizado. Esse problema é mitigado ao utilizar o RVND como tal método de busca local, pois ele é um método de alto nível, no sentido de que ele combina vários algoritmos de busca local com o propósito de obter soluções melhores do que as que resultariam da utilização individual desses algoritmos [27].

1.2 Objetivos

1.2.1 Objetivo geral

Nesse trabalho, objetivamos desenvolver uma abordagem heurística a fim de solucionar o problema de escalonamento do tipo *flow shop* de permutação, tendo a soma dos tempos de conclusão das tarefas, C_{sum}, como função custo.

1.2.2 Objetivos específicos

- Levantar o estado da arte das soluções para o problema de escalonamento.
- Desenvolver um algoritmo heurístico híbrido baseado nas meta-heurísticas ILS e RVND.

1.3 Estrutura do trabalho

A dissertação está dividida em seis seções:

- **Fundamentação Teórica**: Apresenta os principais conceitos encontrados na literatura a fim de facilitar a compreensão e elaboração do trabalho em questão.
- **Trabalhos Relacionados**: São apresentados os trabalhos realizados anteriormente e atualmente apresentados na literatura, a fim de servir como base para o trabalho.
- **FILS**: Apresenta as etapas existentes na pesquisa, descrevendo a estrutura da solução desde a ideação até a apresentação da solução proposta.
- **Apresentação e análise dos resultados**: Apresenta os resultados da solução desenvolvida através de calibração e comparações com outras soluções presentes na literatura para o mesmo tipo de problema.
- **Conclusões**: Conclusões do trabalho baseadas nos testes executados e levantamento da qualidade de solução, assim como propostas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentados os conceitos gerais para nivelamento de conhecimento voltado a área em questão, ou seja, os principais elementos que envolvem este trabalho.

2.1 Função Linear

A função linear é um tipo especial de função do 1° grau cuja lei de formação é do tipo $f(x) = ax$ (a é real e diferente de zero)[68].

Partindo da formatação $f(x) = ax + b$, temos $b \neq 0$, classificando assim a função como sendo linear.

2.2 Programação Linear

A Programação Linear (PL) é uma ferramenta que permite resolver uma série de problemas de otimização em que a função objetivo é uma função linear e as restrições são inequações do tipo $f(x) \leq k$ ($\geq$ ou $=$), onde f é uma função linear e $k \in \mathbb{R}$ [2].

Os problemas de otimização podem ser de vários tipos, dentre eles, temos: encontrar o menor caminho, árvore geradora de menor custo, entre outras. Portanto, a programação linear veio para resolver estes problemas computacionalmente através da solução de equações e inequações características desses problemas.

Para isto, são definidas as variáveis as quais os valores serão determinados. Dependendo do tipo do problema, sendo ele de maximizar ou minimizar algo, insere-se a função objetivo, ou seja, uma expressão linear envolvendo as variáveis previamente determinadas.

Problemas desse tipo também envolvem uma série de restrições, como por exemplo: o caminho deve ser minimizado considerando que não se pode passar de uma casa para outra com mais de 5km de distância, ou seja, a variável da distância que poderia ser percorrida entre duas casas seria menor ou igual a 5km.

Estes problemas podem ser de uma ou mais variáveis e geralmente se utiliza de programação linear para resolvê-los de forma rápida e eficiente, a fim de se obter os melhores resultados.

2.3 Programação Linear Inteira

A Programação Linear Inteira (PLI) caracteriza-se como sendo um problema de Programação Linear (PL) em que todas ou algumas variáveis são discretas, ou seja, assumem apenas valores inteiros [73].

Existem dois tipos de PLI, a Programação Linear Inteira Pura (PLIP), que é quando se tem todas as variáveis sujeitas a condição de integralidade, e a Programação Linear Inteira Mista (PLIM), que é quando apenas algumas das variáveis estão sujeitas a essa condição.

2.4 Heurística

Considera-se um algoritmo como método heurístico quando não há conhecimentos matemáticos completos sobre seu comportamento, ou seja, quando, sem oferecer garantias, o algoritmo objetiva resolver problemas complexos utilizando uma ideia guia, chamada de heurística, e uma quantidade razoável de recursos, como consumo de tempo, para encontrar soluções satisfatórias.[83]

Geralmente as heurísticas são aplicadas em problemas do tipo NP-difícil, onde não são conhecidos métodos que forneçam garantias, provendo assim resultados de bom desempenho.

2.4.1 Heurísticas construtivas

As heurísticas podem ser classificadas em construtivas quando acrescentam componentes individuais à solução inicial até a obtenção de uma solução factível.

2.4.2 Heurísticas de refinamento

Também chamada heurística de busca local, tem como estratégia a realizar modificações em uma solução inicial viável, de forma a tentar melhorá-la, utilizando o conceito de vizinhança.

Através de estudos ao longo dos anos sobre o desempenho dos métodos heurísticos, foram elaboradas estratégias genéricas para a construção de heurísticas, sendo estas estratégias chamadas de meta-heurísticas.

2.5 Meta-heurística

Meta-heurística é um conjunto de conceitos que pode ser utilizado para definir métodos heurísticos aplicáveis a um extenso conjunto de diferentes problemas, ou seja, uma estrutura algorítmica geral que pode ser aplicada a diferentes problemas de otimização com relativamente poucas modificações que possam adaptá-la a um problema específico [83]. Os exemplos mais conhecidos de meta-heurísticas são: algoritmos evolutivos, algoritmos genéticos, GRASP, ant colony optimization, busca tabu, iterated local search, simulated annealing, etc.

As meta-heurísticas fornecem ideias que podem ser aplicadas aos problemas de otimização para os quais não são conhecidos algoritmos específicos eficientes, sejam estes heurísticos ou não.

De acordo com os autores Blum e Roli [17], as meta-heurísticas podem ser classificadas de formas distintas, dependendo das caracteríısticas selecionadas para diferenciá-las. Os autores apresentam algumas maneiras de classificá-las:

- **Baseadas ou não na natureza**: meta-heurísticas baseadas na natureza são aquelas que utilizam metodologias para geração de soluções baseadas em comportamentos da natureza e/ou dos seres-vivos, tais como: seleção natural, mutação, colônia de formigas, etc. Caso contrário, são não-baseadas na natureza.
- **Baseadas em população ou em indivíduos**: depende do número de soluções utilizadas ao mesmo tempo para a geração de novas soluções. Algoritmos que atuam sobre uma única solução são chamados de métodos de trajetória, onde se enquadram as meta-heurísticas baseadas em busca local. Já as meta-heurísticas baseadas em população desempenham um processo de busca que descrevem a evolução de um conjunto de pontos no espaço de busca.
- **Com função objetivo estática ou dinâmica**: depende de como a meta-heurística faz uso da função objetivo. Algumas modificam a função utilizando de informações adquiridas durante o processo de busca para escapar de ótimos locais.
- **Uma ou várias estruturas de vizinhança**: considera o número de vizinhanças utilizadas durante o procedimento de busca local.

2.6 Escalonamento

Seja $J = \{1, ..., n\}$ o conjunto de tarefas que devem ser executadas em um conjunto $M = \{1, ..., m\}$ de máquinas paralelas. Assumindo que cada máquina pode processar apenas uma tarefa por vez e que cada tarefa pode ser processada apenas por uma máquina por vez, várias características relacionadas às máquinas, tarefas e ao processo de atribuição de tarefas às máquinas podem surgir. Desse modo, os problemas de *scheduling* podem apresentar inúmeras particularidades, que quando combinadas, geram um número muito grande de variantes. De maneira geral, nesses problemas se deseja alocar determinadas tarefas a determinados recursos. Nesta categoria, pode-se encontrar problemas como: *crew scheduling*, *timetabling*, *workforce scheduling*, *project scheduling*, *production scheduling*, entre outros [40].

O problema de escalonamento (*scheduling problem*)pode ser classificado de acordo com a nomenclatura proposta por Graham et al [28] composta por três campos ($\alpha \parallel \beta \parallel \gamma$),

onde (α) representa a configuração das máquinas, (β) representa as características das tarefas e (γ) a função objetivo para ser maximizada / minimizada.

2.6.1 Dados

Para cada tarefa $j \in J = \{1, ..., n\}$, os seguintes dados podem ser especificados:

- Número de operações m_j;
- Tempo de processamento p_j, requirido pela tarefa j, caso não varie de máquina para máquina, ou p_j^k, $\forall\ k \in M = \{1, ..., m\}$ caso varie de máquina para máquina;
- Data de liberação r_j a partir da qual j pode começar a ser processada;
- Data de entrega d_j, na qual j deve ser finalizada;
- Pesos w_j e/ou w'_j que indicam a importância relativa da tarefa j;
- Função de custo $f(C_j)$ que mede o custo relacionado ao tempo de término das tarefas;

2.6.2 Configuração das máquinas (α)

- 1: uma máquina – É o caso mais simples dentre as configurações das máquinas, onde as tarefas devem ser processadas em apenas uma máquina;
- P: máquinas paralelas idênticas – Nesse caso, m máquinas paralelas idênticas estão disponíveis para processar as n tarefas J_j. Nessa configuração, as tarefas possuem tempos de processamentos análogos em todas as máquinas;
- Q: máquinas paralelas uniformes – Existem m máquinas em paralelo, porém, com diferentes fatores de velocidade q_k para cada máquina k. Sendo assim, o tempo de processamento p_j^k da tarefa j na máquina k depende desse fator de velocidade;
- R: máquinas paralelas não-relacionadas – Nesse caso, m máquinas em paralelo estão disponíveis para processar as n tarefas j, as quais possuem tempos distintos em cada máquina;
- O: *open shop* – Nessa configuração, as n tarefas j devem ser processadas em todas as m máquinas k por p_j^k unidades de tempo. Não há restrição quanto a sequência de execução das tarefas;
- J: *job shop* – Nesse caso, cada tarefa j possui sua rota predefinida a ser seguida, geralmente distinta das demais;

- F: $flow\ shop$ – Cada tarefa j deve ser processada em cada uma das m máquinas dispostas em série, de forma que todas as tarefas seguem a mesma sequência, ou seja, primeiramente pela máquina m, em seguida a máquina $m+1$, e assim sucessivamente;

2.6.3 Características das tarefas (β)

- *pmtn*: preempção – Nesse caso, a tarefa pode ser dividida em várias etapas, podendo ser interrompida e posteriormente retomada na mesma ou em outra máquina;

- *prec*: precedência – Quando uma tarefa requer o término de outra para ser executada. Podem ser representadas como um grafo acíclico e orientado $G = (V, A)$, onde $V = \{1, ..., n\}$ corresponde as tarefas e $(i, j) \in A$ indica que a tarefa j pode ser iniciada após a conclusão da tarefa i;

- s_{ij}^k : tempo de *setup* – O tempo de *setup* representa uma necessidade de preparação entre o processamento de duas tarefas i e j, variando de acordo com a ordem de execução das tarefas ou serem independentes de sequência (s_j), quando apenas a tarefa processada define o tempo de preparação. Em alguns casos, esse tempo pode variar também em função da máquina, o que pode ser representado por s_{ij}^k;

- r_j: datas de liberação – No caso das tarefas começarem a ser processadas em tempos diferentes, estas devem ser especificadas. Se $r_j = 0, \forall\ j \in J$, então a indicação r_j não deve aparecer no campo β;

2.6.4 Função objetivo (γ)

O critério de otimalidade está diretamente relacionado com o momento em que o processamento da tarefa j é finalizado, chamado de *completion time*, denotado por C_j.

- C_{max}: Refere-se à minimização do *makespan*, que é definido como sendo o máximo dos tempos de término dentro todas as tarefas $j \in J$. Com isso, visa-se minimizar o tempo de término da última tarefa a ser finalizada;

- L_{max}: O atraso (*lateness*) de uma tarefa é obtido entre o tempo em que uma tarefa j é finalizada e seu tempo de término desejado, ou seja, $L_j = C_j - d_j$, podendo ser positivo ou negativo. O objetivo é minimizar o atraso máximo $max(L_1, ..., L_n)$;

- $\Sigma\ w_j C_j$: A minimização da soma ponderada dos tempos de término das tarefas, onde cada tarefa j possui um peso w_j associado;

- $\Sigma\ w_j F_j$, $\Sigma\ F_j$: O *flowtime* é definido por $F_j = C_j - r_j$. O objetivo é minimizar o *flowtime* total ponderado (denotado por $\Sigma w_j F_j$) ou o *flowtime* total não-ponderado

(denotado por ΣF_j). Quando todas as datas de liberação são $r_j = 0$, o critério de *flowtime* é equivalente ao critério de tempo de conclusão [72];

- $\Sigma\ w_j T_j$: A minimização da soma dos atrasos ponderados das tarefas (*weighted tardiness*). O *tardiness*, diferentemente do *lateness* pode assumir apenas valores positivos. Desta maneira, o *tardiness* pode ser definido como sendo $T_j = max(L_j, 0)$;

- $\Sigma\ (w_j^{'}\ E_j + w_j^{'}\ T_j)$: A minimização da soma dos atrasos e antecipações ponderados das tarefas (*weighted earliness and tardiness*). De maneira análoga ao atraso, a antecipação é obtida pela diferença entre o tempo desejado do término da tarefa j e o tempo em que ela será finalizada, ou seja, $E_j = max(d_j - C_j, 0)$.

2.7 Formulação Matemática

São dados um conjunto de tarefas $J = \{J_1, \ldots, J_n\}$ e um conjunto de máquinas $M = \{M_1, \ldots, M_m\}$. Cada tarefa J_j deve ser processada em todas as máquinas em M; portanto, J_j compreende um conjunto de m operações, $O_j = \{O_{1j}, \ldots, O_{mj}\}$. Cada operação O_{ij} tem um tempo de processamento associado, p_{ij}, que denota quanto tempo a máquina M_i gasta para processar a tarefa J_j. Cada máquina pode processar no máximo uma tarefa por vez, e cada tarefa pode ser processada em no máximo uma máquina por vez. A preempção não é permitida; isto é, uma vez escalonada em uma máquina, a tarefa deve ser completamente processada. Além disso, a ordem em que cada máquina processa as tarefas é a mesma para todas as máquinas.

Para uma dada sequência de tarefas $\pi = (\pi_1, \ldots, \pi_n)$, o tempo de conclusão da tarefa π_j na máquina M_i é calculado de forma recursiva:

$$C_{ij} = \begin{cases} \max\{C_{i-1,j}, C_{i,j-1}\} + p_{ij}, & 1 \leq i \leq m, 1 \leq j \leq n \\ 0, & i = 0 \\ 0, & j = 0 \end{cases} . \tag{1}$$

O tempo de conclusão da tarefa π_j é definido como sendo o seu tempo de conclusão na última máquina,

$$C_j = C_{mj}. \tag{2}$$

O problema é achar uma sequência de escalonamento que minimiza a soma dos tempos de conclusão de todas as n tarefas,

$$C_{sum} = \sum_{j=1}^{n} C_j. \tag{3}$$

3 TRABALHOS RELACIONADOS

3.1 Revisão da literatura

Será apresentado, em ordem cronológica, a maioria dos trabalhos relacionados a máquinas paralelas idênticas na Tabela 1, máquinas paralelas uniformes na Tabela 2 e máquinas paralelas não-relacionadas na Tabela 3, onde são expostos, para cada trabalho, o método de resolução, características do problema, incluindo tempos de *setup* (s_{ijk}), datas de liberação das tarefas diferentes de zero (r_j) e tempo de ociosidade entre as tarefas (IT). Também o objetivo a ser minimizado, especificamente, soma dos atrasos (T_j), soma das antecipações e atrasos ($E_j + T_j$), soma do tempo de fluxo (F_j), ou seja, todo o passo a passo do *job* e soma dos tempos de término das tarefas (C_j), sendo estas ponderadas ou não [40].

Tabela 1: Máquinas paralelas idênticas

Referência	**Métodos**	s_{ijk}	r_j	IT	T_j	$E_j + T_j$	F_j	C_j
Webster (1992) [91]	*Lower bounds*		✓				✓	
Webster (1993) [92]	Regras ótimas de prioridade para um caso especial do problema $P\|\ R_j\|\ \Sigma\ w_j\ F_j$		✓				✓	
Webster (1995) [93]	*Lower bounds*		✓				✓	
Belouadah e Potts (1994) [13]	B&B + RL							✓
Lee e Pinedo (1997) [42]	Fase de pré-processamento + *Apparent Tardiness Cost with Setup* ($ATCS$) + SA	✓			✓			
Koulamas (1997) [39]	*Lower bounds*, SA				✓			
Azizoglu e Kirca (1998) [8]	B&B				✓			
Azizoglu e Kirca (1999) [7]	B&B							✓
Chen e Powell (1999) [18]	GC							✓

continua na próxima página

Referência	**Métodos**	s_{ijk}	r_j	IT	T_j	$E_j + T_j$	F_j	C_j
Radhakrishnan e Ventura (2000) [66]	SA	✓				✓		
Eom et al (2002) [21]	EDD + ATCS + TS	✓			✓			
Yalaoui e Chu (2002) [99]	B&B				✓			
Sun e Wang (2003) [84]	PD, Heurística construtiva					✓		
Kim et al (2006) [38]	TS	✓	✓	✓	✓			
Omar e Teo (2006) [59]	Formulação PIM	✓		✓		✓		
Yalaoui e Chu (2006) [98]	B&B		✓	✓				✓
Shim e Kim (2007b) [76]	B&B				✓			
Kedad-Sidhoum et al (2008) [37]	Formulação indexada no tempo, RL, GC, Heurística		✓	✓		✓		
Feng e Lau (2008) [25]	*Squeaky Wheel Optimization* (*SWO*)	✓		✓		✓		
Rios-Solis e Sourd (2008) [70]	*Exponential Neighborhood Search*					✓		
Pfund et al (2008) [64]	Heurística *Apparent Tardiness Cost with Setup and Ready times* (*ATCSR*)	✓	✓	✓	✓			
Nessah et al (2008) [57]	B&B		✓	✓				✓
Biskup et al (2008) [16]	Heurística construtiva				✓			
Rodrigues et al (2008) [71]	ILS				✓			
Tanaka e Araki (2008a) [87]	B&B + RL				✓			

continua na próxima página

Referência	Métodos	s_{ijk}	r_j	IT	T_j	$E_j + T_j$	F_j	C_j
Baptiste et al (2008) [12]	*Lower bounds*		✓	✓	✓			✓
Mason et al (2009) [54]	Heurística de mover blocos			✓		✓		
Pessoa et al (2010) [63]	Formulação indexada no tempo, $Branch - cut - and - price$				✓			
Jouglet e Savourey (2011) [35]	B&B		✓	✓	✓			
M'Hallah e Al-Khamis (2012) [56]	PIM, Heurística híbrida (*steepest descent* + GA + SA)			✓		✓		
Croce et al (2012) [20]	ILS + *Very Large Neighborhood Search*				✓			
Amorim (2013) [3]	GA + PR + ILS					✓		
Amorim et al (2013) [4]	GA Híbrido					✓		
Schaller (2014) [74]	TSs, GAs, B&B	✓			✓			
Xi et al (2015) [95]	Heurística construtiva $look - ahead$	✓	✓	✓	✓			

Fonte: Kramer, 2015 [40]

Tabela 2: Máquinas paralelas uniformes

Referência	Métodos	s_{ijk}	r_j	IT	T_j	$E_j + T_j$	F_j	C_j
Webster (1992) [91]	*Lower bounds*		✓				✓	
Guinet (1995) [29]	SA				✓			
Azizoglu e Kirca (1998) [8]	B&B				✓			
Azizoglu e Kirca (1998) [7]	B&B							✓

continua na próxima página

Referência	Métodos	s_{ijk}	r_j	IT	T_j	$E_j + T_j$	F_j	C_j
Chen e Powell (1999) [18]	CG							✓
Sivrikaya-Serifoglu e Ulusoy (1999) [78]	GA	✓	✓	✓		✓		
Balakrishnan et al (1999) [9]	Formulação matemática + Decomposição de Benders	✓	✓			✓		
Biskup e Feldmann (2001) [15]	Proposição de instâncias							✓
Bilge et al (2004) [14]	TS	✓	✓		✓			
Anghinolfi e Paolucci (2007) [5]	TS+SA+VNS	✓	✓		✓			
Armentano e de França Filho (2007) [6]	GRASP	✓	✓		✓			
Raja et al (2008) [67]	SA + Lógica *fuzzy*	✓				✓		
Yousefi e Yusuff (2013) [101]	*Imperialist Competitive Algorithm*			✓		✓		✓
Lin (2013) [48]	Modelos / Formulações			✓				✓
Li et al (2014) [44]	*Agent − based algorithm + Lower bounds*		✓	✓				✓

Fonte: Kramer, 2015 [40]

Tabela 3: Máquinas paralelas não-relacionadas

Referência	Métodos	s_{ijk}	r_j	IT	T_j	$E_j + T_j$	F_j	C_j
Webster (1992) [91]	*Lower bounds*		✓				✓	
Zhu e Heady (2000) [103]	Formulação PIM	✓				✓		

continua na próxima página

Referência	**Métodos**	s_{ijk}	r_j	IT	T_j	$E_j + T_j$	F_j	C_j
Bank e Werner (2001) [10]	Heurísticas construtivas + iterativas, SA		✓	✓		✓		
Weng et al (2001) [94]	Heurísticas construtivas	✓						✓
Liaw et al (2003) [46]	B&B				✓			
Zhou et al (2007) [102]	ACO				✓			
Shim e Kim (2007a) [75]	B&B				✓			
Logendran et al (2007) [51]	Seis algoritmos baseados em TS	✓	✓		✓			
Akyol e Bayhan (2008) [1]	Rede neural	✓				✓		
Li e lin Yang (2009) [45]	*Survey*	✓	✓			✓		
Vallada e Ruiz (2012) [89]	Formulação PIM, GA	✓		✓		✓		
Lee et al (2013) [41]	TS	✓			✓			
Polyakovskiy e M'Hallah (2014) [65]	Heurística $Multi - Agent\ System$			✓		✓		
Nogueira et al (2014) [58]	GRASP + RC + ILS	✓		✓		✓		
Lin e Hsieh (2014) [49]	ATCSR modificada e $Electromagnetism - like\ Algorithm$ (EMA)	✓	✓		✓			
Şen e Bülbül (2015) [104]	Relaxação baseada em preempção + Decomposição de Benders		✓			✓	✓	

Fonte: Kramer, 2015 [40]

Para resolução destes problemas envolvendo máquinas paralelas, temos as seguintes heurísticas:

- `Heurísticas baseadas em inteligência artificial`: Akyol e Bayhan (2008)

[1], Raja et al (2008), Polyakovskiy e M'Hallah (2014) [65] [67], Li et al (2014) [44].

- `Heurísticas construtivas ou de refinamento`: Lee e Pinedo (1997) [42], Bank e Werner (2001) [10], Weng et al (2001) [94], Sun e Wang (2003) [84], Feng e Lau (2008) [25], Biskup et al (2008) [16], Pfund et al (2008) [64], Mason et al (2009) [54], Xi et al (2015) [95].
- `Meta-heurísticas baseadas em busca local`:
  - `GRASP`: Armentano e de França Filho (2007) [6].
  - `ILS`: Rodrigues et al (2008) [71].
  - *`Large Neighborhood Search`*`(LNS)`: Rios-Solis e Sourd (2008) [70].
  - `SA`: Guinet (1995) [29], Lee e Pinedo (1997) [42], Koulamas (1997) [39], Radhakrishnan e Ventura (2000) [66], Bank e Werner (2001) [10].
  - `TS`: Eom et al (2002) [21], Bilge et al (2004) [14], Kim et al (2006) [38], Logendran et al (2007) [51], Lee et al (2013) [41], Schaller (2014) [74].
- `Meta-heurísticas baseadas em população`:
  - `ACO`: Zhou et al (2007) [102].
  - `EA`: Yousefi e Yusuff (2013) [101], Lin e Hsieh (2014) [49].
  - `GA`: Sivrikaya-Serifoglu e Ulusoy (1999) [78], Vallada e Ruiz (2012) [89], Amorim (2013) [3], Amorim et al (2013) [4], Schaller (2014) [74].
- `Meta-heurísticas híbridas`: Anghinolfi e Paolucci (2007), Croce et al (2012) [20], M'Hallah e Al-Khamis (2012) [56], Nogueira et al (2014) [58].

Também temos os seguintes métodos exatos:

- `B&B + RL`: Belouadah e Potts (1994) [13], Tanaka e Araki (2008a) [87].
- `B&B baseados em regras de dominâncias`: Azizoglu e Kirca (1998) [8] (1999) [7], Yalaoui e Chu (2002) [99], Liaw et al (2003) [46], Yalaoui e Chu (2006) [98], Shim e Kim (2007a) [75] (2007b) [76], Jouglet e Savourey (2011) [35], Nessah et al (2008) [57], Schaller (2014) [74].
- `B&B baseados em relaxação linear, em CG e/ou planos de corte`: Chen e Powell (1999) [18], Pessoa et al (2010) [63].
- `Formulações compactas baseadas em PIM`: Balakrishnan et al (1999) [9], Zhu e Heady (2000) [103], Omar e Teo (2006) [59], Vallada e Ruiz (2012) [89], Lin (2013) [48].

- *Lower bounds* `baseados em RL e GC`: Kedad-Sidhoum et al (2008) [37].
- `Relaxação baseada em preempção`: Şen e Bülbül [104].

No trabalho de Fanjul-Peyro et al [22], os autores analisaram um problema de programação de máquinas paralelas em que o processamento de tarefas nas máquinas requer um número de unidades de um recurso escasso. Esse número depende da tarefa e da máquina. A disponibilidade de recursos é limitada e fixa ao longo do horizonte de produção. O objetivo considerado é a minimização do *makespan*.

O problema foi modelado por meio de dois problemas de programação linear inteira. Um deles é baseado em um modelo previamente proposto na literatura. O outro, baseado na semelhança com os problemas de empacotamento. O segundo é uma contribuição original do artigo. Como os modelos apresentados são incapazes de resolver instâncias de tamanho médio para otimalidade, são propostos três estratégias matemáticas para cada um desses dois modelos. Os algoritmos propostos são testados em uma extensa experiência computacional e os resultados mostraram que as estratégias matemáticas superam significativamente os modelos matemáticos.

- `Problema UPM` - *Unrelated parallel machine scheduling problem*, para minimizar o *makespan* (tempo máximo de conclusão da tarefa).

  Fanjul-Peyro et al considera problemas acadêmicos e problemas do setor de produção, principal foco. Então o trabalho considera o UPM com número de recursos fixos para processar as tarefas. Estes recursos são *renováveis* estando disponíveis após cada processamento e *discretos* por ser um número inteiro positivo.

- `Problema UPMR` - *Dynamic unrelated parallel machine scheduling problem with additional resources.*

  Não especificado, porque não há atribuição de máquina pré-fixado, e *dinâmico* no sentido de que a alocação de recursos para as máquinas não precisa ser fixo em todo horizonte de tempo.

  A sua versão estática assume que a alocação de recursos para a máquina é obtida e fixa por todo o horizonte de tempo.

  É feito o uso de dois diferentes MILPs: Um baseado no modelo de pesquisas anteriores introduzidas na literatura chamado de *MILP program for the UPMR* (UPMR-S), e o segundo modela um problema UPMR como um problema especial de empacotamento (*packing*)(principal contribuição teórica do trabalho pela originalidade) chamado de *UPMR problem model based on strip-packing* (UPMR-P), o qual utiliza de variáveis binárias.

As meta-heurísticas utilizadas foram:

- *Machine-assignment fixing*: Atribui tarefas às máquinas resolvendo o problema da UPM, para depois resolver o problema da UPMR, no qual a atribuição da máquina de trabalho é aquela dada pela solução UPM. Em seguida, resolve-se o UPMR com uma atribuição desse tipo. Usa-se essa estratégia porque tanto a UPM quanto o problema especificado da UPMR são muito mais rápidos de resolver do que a UPMR não especificada.
- *Job-machine reduction*: Reduz o conjunto de possíveis atribuições da máquina de trabalho, descartando, para cada tarefa, as máquinas que geram os maiores tempos de processamento. Essa estratégia foi escolhida devido aos bons resultados obtidos na UPM (Fanjul-Peyro e Ruiz (2011)[23])
- *Greedy-based fixing*: É uma estratégia gulosa, na qual o problema é sequencialmente resolvido para pequenos subconjuntos de tarefas, mantendo as atribuições encontradas nas iterações anteriores fixadas. Essa estratégia foi escolhida tanto por sua simplicidade quanto pelo amplo e principalmente bem-sucedido uso que os algoritmos gulosos e suas variantes tiveram na literatura de programação.

O modelo UPMR-S obtém soluções mais otimizadas do que o modelo UPMR-P, mas ao mesmo tempo também possui um grande número de soluções ruins. O modelo UPMR-P sempre fornece uma solução, que é um traço muito desejado. No que diz respeito à matemática, os únicos que nem sempre são capazes de encontrar soluções são MAF-S e JMR-S, ambos baseados em UPMR-S. De particular interesse é o MAF-P, que é capaz de encontrar soluções que mais tarde provaram ser ótimas em quase 50% dos casos. Neste cenário, é difícil comparar métodos quando nem todos eles deram soluções para todas as instâncias.

Por fim, utilizando de RPD *(Relative percentage deviation)*, os autores compraram alguns dos resultados gerados em testes e concluíram que, cada um dos três algoritmos matemáticos obtidos a partir do modelo UPMR-P são superiores aos seus equivalentes correspondentes obtidos do modelo UPMR-S, o que é outra indicação de que o modelo baseado em empacotamento é melhor.

No trabalho de Xu et al (2018) [97], é utilizada restrição de disponibilidade em uma das duas máquinas utilizadas. Sete formulações de programação inteira mista (MIPFs) são propostas para o problema em que a restrição de disponibilidade é imposta na primeira máquina. Outros sete análogos são propostos para a segunda máquina. Resultados numéricos indicam que, para qualquer um dos dois problemas, cada um dos três primeiros MIPFs correspondentes podem resolver instâncias de tamanho de até 100 "jobs"(ou tarefas) em tempos razoáveis.

Existem portanto os seguintes MPIFs:

- `MIPF baseada em` $Wagner's\ idea$: Se aplica a tecnologia *one-job-one-position* e utiliza o tempo de ociosidade entre duas tarefas (IT) consecutivas e os tempos de espera (T_j) das tarefas são conectivos ou intermediários para formular as restrições.
- `MIPF baseada em` $Wilson's\ idea$: Similar a $Wagner's\ idea$ exceto que adota variáveis para apresentar o tempo de início das tarefas (r_j) ao invés de variáveis de tempo ocioso (IT) e de espera (T_j) entre as tarefas.
- `MIPF baseada em` $Tseng$ `e` $Standfford's\ idea$: Similar a $Wilson's\ idea$ exceto por adotar variáveis para apresentar o tempo de conclusão das tarefas C_j ao invés de tempo de início.
- `MIPF baseada em` $Manne's\ idea$: É baseada na observação de que deve haver uma ordem entre duas tarefas em qualquer sequência de processamento viável.
- `MIPF baseada em` $Liao$ `e` $You's\ idea$: É similar ao $Manne's\ idea$ exceto por introduzir variáveis adicionais como média para reformular conjuntos de restrições.

Através de combinações, os autores geram comparativos para cada combinação e, por fim, concluem que as três primeiras MIPFs são capazes de resolver um conjunto de 100 tarefas em 1 hora.

As instâncias mais utilizadas na literatura são as de Taillard [86] desde 1993. Elas são divididas em n *jobs* em m máquinas, também com os respectivos problemas, sendo eles *Flow shop sequencing*, *Job shop scheduling* e *Open shop scheduling*. Em seu site (http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html) existe também uma área para cada grupo de instâncias com os *lower* e *upper bounds*, comparando diversos tipos de soluções para instâncias, porém, sem diferenciar que estratégia foi utilizada, ou seja, é indiferente se o autor utilizou meta-heurística ou programação linear inteira.

Um exemplo de trabalho que adota as instâncias de Taillard como recursos para testes é o artigo produzido na Universidade Federal do Rio Grande do Norte, de título *New VNS heuristic for total flowtime flowshop scheduling problem* [19], onde os autores propõem uma nova heurística VNS (*Variable Neighborhood Search*) [55]. Os passos para construção da solução consistem em:

- Uma solução inicial;
- Um esquema de perturbação, chamado de procedimento de perturbação (*shake*), utilizado para escapar de um ótimo local comum em diversos vizinhos;

- Uma conjunto de vizinhos para busca local;
- Um esquema para definir quando mudará de vizinhança.

Porém, os autores elaboram quatro modelos diferentes de VNS, chamando-os de VNS1, VNS2, VNS3, VNS4, VNS5 e VNS6. Após alguns testes, é definido que o algoritmo mais eficiente é o VNS4, o qual será considerado neste trabalho, pois trata o mesmo tipo de problema e, consequentemente, serão utilizadas as mesmas instâncias de Taillard [86] para comparação.

Os autores utilizam também outros dois algoritmos para comparação de resultados, são eles o AGA e V4AGA: *Asynchronous Genetic local search Algorithm* [96]. Com estes, o autor utiliza mais algumas instâncias, as de Kruskal-Wallis, que não serão consideradas neste trabalho.

4 FILS

Neste capítulo será discutida a metodologia escolhida para a resolução do problema em questão. A avaliação de eficiência será baseada no escalonamento de instruções em mais de um processador sem perdas devido a atrasos.

Com base nas informações anteriores, torna-se necessário utilizar algoritmos híbridos para solucionar o problema de escalonamento de instruções em múltiplos processadores, sendo assim uma metodologia formal.

4.1 Metodologia para solução do problema

Devido aos problemas tratados pertencerem a classe NP-Difícil, a sua resolução por meio de métodos exatos exige um tempo computacional de ordem exponencial em relação ao tamanho do problema. Usualmente, os métodos exatos não obtém êxito na resolução de problemas de dimensões elevadas em tempo computacional aceitável [40]. Para esses casos, heurísticas e meta-heurísticas são efetivas apesar de não garantirem a solução ótima do problema, tais métodos retornam soluções viáveis e sub-ótimas em tempo computacional aceitável.

A metodologia a ser empregada neste trabalho consiste em uma abordagem híbrida baseada na meta-heurística *Iterated Local Search* (ILS) [52] e *Random Variable Neighborhood Descent* (RVND) [55]. O ILS consiste em um processo de geração de solução inicial, busca local, perturbação e critério de aceitação, demonstrado a seguir no Algoritmo 1:

Algoritmo 1 ILS

```
1: função ILS
2:     π_0 ← GeraSolucaoInicial
3:     π ← BuscaLocal(π_0)
4:     enquanto Critério de parada não for atendido faça
5:         π' ← Pertubacao(π)
6:         π'' ← BuscaLocal(π')
7:         π ← CriterioAceitacao(π, π'')
8:     fim enquanto
9: fim função
```

Fonte: Lourenço et al (2003) [52]

Devido à quantidade de algoritmos de busca local que ele utiliza de forma aleatória, o RVND tem um comportamento altamente *intensificador* e, portanto, tende a ficar preso em ótimos locais. Já o ILS, com a sua fase de perturbação, tem um comportamento altamente *diversificador*. Juntos, esses dois métodos são complementares e tendem a dar melhores resultados do que quando utilizados separadamente [27].

4.2 Representação das entradas e saídas

A heurística FILS tem os seguintes parâmetros:

- m: a quantidade de máquinas $(M_1, M_2, \ldots, M_m)$;
- n: a quantidade de tarefas $(J_1, J_2, \ldots, J_n)$; e
- p_{ij}: uma função que retorna o tempo de processamento da tarefa J_j na máquina M_i.

Essas informações podem ser codificadas numa matriz de números inteiros, p, cuja ordem é $m \times n$; nessa representação, a ordem da matriz dá, respectivamente, as quantidades de tarefas e máquinas, e a célula p_{ij} dá o tempo de processamento da tarefa J_j na máquina M_i.

Além dessas entradas, FILS recebe t_{max}, o tempo máximo, em segundos, que o algoritmo é permitido executar.

Com relação à solução, a heurística codifica-a num vetor de números inteiros, s, cuja dimensão é n. Esse vetor representa a ordem de processamento das tarefas; e.g., numa instância com 5 tarefas, se a heurística retorna o vetor $s = [3, 1, 2, 5, 4]$, significa que o escalonamento resultante é $\pi = (J_3, J_1, J_2, J_5, J_4)$.

4.3 Geração da Solução Inicial

O processo de geração da solução inicial consiste no estudo de uma heurística comumente utilizada no ILS.

Quan-Ke e Ruiz, 2012 [60], utilizaram, para gerar a solução inicial, o algoritmo LR(n/m), desenvolvido por Liu e Reeves, 2001 [50], onde a heurística gera $\lfloor n/m \rfloor$ sequências de escalonamento e retorna a sequência que minimiza C_{sum}. A heurística considera como S uma sequência parcial que contém os *jobs* já agendados, e U como um conjunto de *jobs* não agendados. Ao selecionar um *job* em U para anexar em S, utiliza-se uma função de índice para comparar os *jobs* em U. A função de índice considera o *weighted total machine idle time* e o *artificial total flow time*, onde IT_{jk} estima o tempo ocioso induzido pelo escalonamento da tarefa J_j na posição $k+1$ e AT_{jk} estima o impacto desse escalonamento nos tempos de conclusão das tarefas ainda em U.

O algoritmo LR tem complexidade de tempo $O(n^3m)$. Um dos fatores que contribuem para tal complexidade é que a estimativa dada por AT_{jk} baseia-se na geração de uma tarefa artificial, J_a; essa tarefa representa a média das tarefas em $U - \{J_j\}$. A complexidade de tempo para computar o tempo de processamento de J_a, p_a, é $\mathcal{O}(nm)$. Fernandez-Viagas e Framinan, 2015 [26] notam que esse fator não colabora muito para

a qualidade do resultado do algoritmo, e propõem a remoção dessa etapa, em favor da utilização de dois parâmetros, a e b, para ponderar os outros fatores realmente decisivos: o *tempo ocioso induzido pela tarefa recém-inserida*; e o *tempo de conclusão dessa tarefa na máquina* M_m. O algoritmo resultante dessa modificação é chamado de FF(x).

A heurística FF(x) [26] utiliza a ideia presente em LR(n/m) de número decrescente de avaliações de soluções, começando com $n-1$ avaliações e finalizando com 0. Desse modo, a heurística é composta por $n-1$ etapas com um máximo de $n-1$ avaliações. No entanto, ao contrário do LR(n/m), se concentra na avaliação de cada solução tentando reduzir a complexidade do algoritmo. Quando se introduz um novo *job* no final da sequência, considera-se três elementos:

- *Tempo ocioso induzido pelo job recém-inserido*: Esse tempo inativo influencia os próximos *jobs* a serem inseridos. Essa influência diminui a cada etapa (sendo 0 a última etapa). Seu cálculo possui uma complexidade O(m), pois somente o tempo de conclusão do *job* anterior em cada máquina é necessário. Para uma determinada iteração k, esses dados são conhecidos da iteração anterior (ou 0 se for o primeiro *job*).

- *Tempo de conclusão na máquina m do job recém-inserido*: Sua influência no tempo total de fluxo é clara, pois o tempo de conclusão de cada trabalho na máquina m está incluído na função objetivo. Esses dados podem ser calculados dentro de O(m) usando o tempo de conclusão em cada máquina do *job* anterior.

- *Tempo de conclusão na máquina m do job artificial*: Influencia a função objetivo de maneira indireta, pois é um indicador de tempo de conclusão na máquina m, porém, o cálculo desse tempo de conclusão tem uma complexidade de $n.m$.

Como na heurística de LR(n/m), pretende-se que, uma vez que um *job* seja agendado em uma posição, ele permaneça nessa posição, e a escolha da posição adequada de um *job* é fundamental. O problema reside, portanto, na ponderação da influência dos elementos acima mencionados. Para fazer isso, utiliza-se dois parâmetros (a e b) para equilibrar os dois primeiros elementos, ou seja, tempo ocioso e tempo de conclusão do *job* recém-inserido. Por outro lado, exclui-se o terceiro elemento (tempo de conclusão do trabalho artificial), uma vez que sua influência na função objetivo não é tão direta quanto os outros dois elementos, e sua consideração aumentaria a complexidade do algoritmo para n^3m.

Com isso, a heurística FF(x) é a seguinte:

1. Classifica os *jobs* de acordo com uma ordem não-descendente de indicador ξ'_{j0}, quebrando nós em favor de *jobs* com maior IT'_{j0}. Denota-se por I o vetor assim obtido.

2. Obtém-se x sequências parciais $\pi^i(i = 1, ..., x)$ de comprimento 1, onde o primeiro (e único) job da sequência π^i é o job na posição i em I. Armazena-se em U^i os $jobs$ não agendados em π^i.

3. De $k = 2$ a n: Para cada sequência parcial π^i, remove-se de U^i o job para qual o valor mínimo de $\xi'_{j,k}$ é encontrado e coloca-o na última posição de π^i.

4. Retorna-se a sequência (final) π^i produzindo o menor tempo de conclusão.

FF(x) possui complexidade de tempo $\mathcal{O}(n^2m)$ e apresenta resultados melhores do que o LR; portanto, ele será o nosso gerador de solução inicial, mais especificamente na forma FF(n/m). A heurística FF(x) é apresentada no Algoritmo 2.

Algoritmo 2 FF

```
1: função FF(x)
2:     I ← Sort(FFCompJobs, J)          ▷ Ordena as tarefas de forma não-decrescente segundo ξ_j0
3:     π ← ∅
4:     para l ← 1, x faça
5:         U ← J − {I_l}
6:         π' ← (I_l)
7:         para j ← 2, n faça
8:             J_chosen ← FFNextJob(π', U)      ▷ Retorna a tarefa em U com o menor ξ_jk
9:             U ← U − {J_chosen}
10:            π' ← concat(π', J_chosen)
11:        fim para
12:        se π = ∅ ∨ C_sum(π') < C_sum(π) então
13:            π ← π'
14:        fim se
15:    fim para
16:    retorna π
17: fim função
```

Fonte: Fernandez-Viagas e Framinan, 2015 [26]

4.4 Busca local

Aqui serão detalhados os processos que consistem na busca local do algoritmo.

É encontrado na literatura que a combinação da meta-heurística ILS com o método *Random Variable Neighborhood Descent* (RVND) [55] na fase de busca local, permite a obtenção de soluções de alta qualidade para diferentes tipos de problemas (Subramanian et al, 2010 [82]; Silva et al, 2012 [77]; Penna et al, 2013 [62]; Subramanian e Battarra, 2013 [80]; Martinelli et al, 2013 [53]; Subramanian et al, 2014 [81]; Vidal et al, 2015 [90]; Farias et al, 2015 [24]).

O RVND (Algoritmo 3) é uma variação do VND desenvolvido por Mladenovic e

Hansen, 1997 [55], e consiste na seleção aleatória de uma vizinhança não explorada de uma lista de vizinhanças, onde havendo falha por parte da vizinhança em encontrar uma solução de custo melhor que a atual, tal vizinhança é removida da lista. No caso de uma melhor solução ser encontrada em uma vizinhança da lista, a lista é atualizada re-inserindo todas as vizinhanças.

Algoritmo 3 RVND

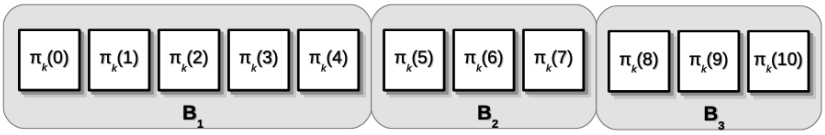
```
1: função RVND(N, π)
2:     N' ← Shuffle(N)
3:     k ← 1
4:     enquanto k ≤ |N'| faça
5:         π' ← BestNeighbor(N'_k, π)
6:         se f(π') < f(π) então        ▷ ou f(π') > f(π), se f é uma função utilidade
7:             π ← π'
8:             N' ← Shuffle(N)
9:             k ← 1
10:        senão
11:            k ← k + 1
12:        fim se
13:    fim enquanto
14:    retorna π
15: fim função
```

Fonte: Mladenovic e Hansen (1997) [55]

4.5 Refinamento da solução

Nesta seção serão apresentadas as estruturas de vizinhanças utilizadas para a fase de intensificação, ou seja, série de sucessivos refinamentos da solução atual. Para isso, utilizaremos o RVND [55] com as estruturas apresentadas a seguir. A fim de ilustrar o funcionamento de tais, serão utilizados blocos e o cálculo de custo de uma solução vizinha π'.

Figura 2: Representação em blocos



Fonte: Kramer, 2015[40]

A Figura 2 ilustra a seguinte sequência: seja $\pi_k = (\pi_k(0), \pi_k(1), \pi_k(2), ..., \pi_k(n_k+1))$ uma sequência de tarefas contendo $n_k + 2$ tarefas ($\pi_k(0)$ e $\pi_k(1)$ são tarefas auxiliares), divididas em B blocos, sendo cada B uma subsequência de tarefas.

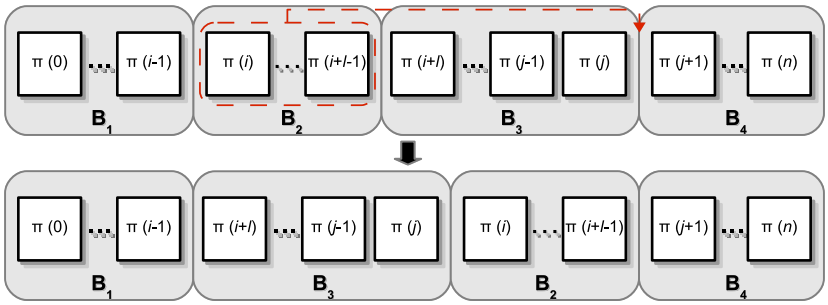
O custo de uma sequência (quando não se permite tempo ocioso) é obtido através da soma dos custos dos B blocos. Na Figura 2, o custo será a soma dos custos de B_1 até B_3.

O número de blocos de uma sequência é dada de acordo com as necessidades das vizinhanças:

4.5.1 *l-Block Insertion*

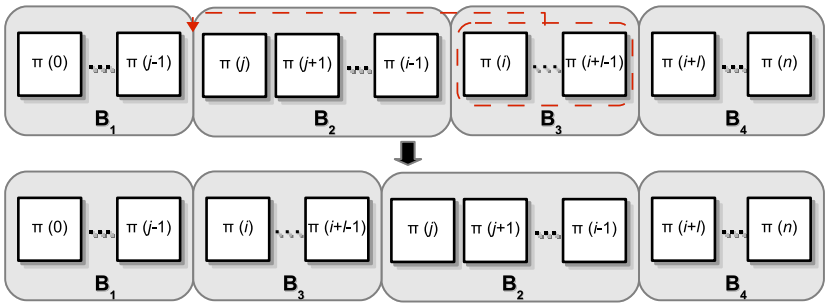
Consiste no deslocamento de um bloco de tamanho l para frente ($i < j$) (Figura 3) ou para trás ($i > j$) (Figura 4), dentro de uma máquina k.

Figura 3: *l-block insertion forward*



Fonte: Kramer, 2015 [40]

Figura 4: *l-block insertion backward*



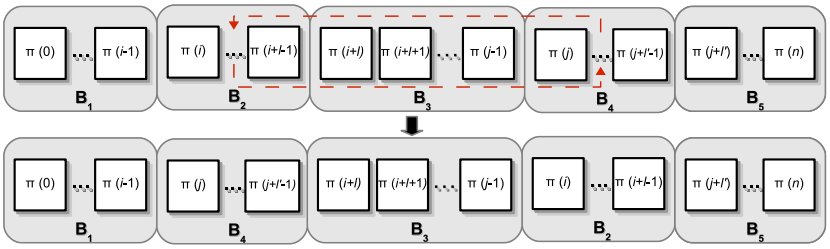
Fonte: Kramer, 2015[40]

Para o problema de *flow shop*, alterar a posição de um *job* significa alterá-lo em todas as máquinas. As figuras acima ilustram o movimento do bloco, porém, o índice k da imagem não se aplica ao problema em questão, pois, no *flow shop*, qualquer esquema de geração de vizinhanças é, simultaneamente, intra e inter-máquinas.

4.5.2 *Block Swap*

Consiste nos movimentos de troca de posições de dois blocos de tamanhos l e l', respectivamente, dentro da mesma máquina k. (Figura 5)

Figura 5: *Block swap*



Fonte: Kramer, 2015[40]

FILS realiza a exploração exaustiva dos espaços de soluções dessas estruturas; i.e., verificamos todas as combinações possíveis, e escolhemos a solução que apresenta melhor resultado. A complexidade de tempo da avaliação de ambas as estruturas de vizinhança é $O(n^2)$.

4.6 Perturbação

O mecanismo responsável pela diversificação das soluções, chamado de perturbação, tem como ideia principal modificar a solução ótima local, porém mantendo algumas características que possibilitem a obtenção de outro ótimo local melhor em menos tempo, uma espécie de histórico de melhorias.

Portanto, foram utilizados dois procedimentos de perturbação:

- *Multiple swap (l, l')*: realizando de 1 até 7 movimentos de troca de posição entre duas subsequências, cujos tamanhos são representados por l e l'.
- *Multiple swap (1, 1)*: realizando um número aleatório entre 1 e 7 trocas de posição. Cada movimento consiste na remoção de duas tarefas aleatórias e reinserção em posições aleatórias.

No ambiente de múltiplas máquinas, os mecanismos apresentados acima são escolhidos de forma aleatória com a mesma probabilidade de escolha. Caso o *Multiple swap (l, l')* seja escolhido e falhe na perturbação da solução, o *Multiple swap (1, 1)* é executado.

O algoritmo de perturbação (*FilsPertubation*) é apresentado no Algoritmo 4.

Algoritmo 4 FILSPERTUBATION

```
1: função FilsPertubation(π)
2:     se RandNth({0,1}) = 0 então
3:         π' ← MultipleSwap(l, l', π)
4:         se π' = π então                          ▷ π' não foi perturbada
5:             π' ← MultipleSwap(1, 1, π)
6:         fim se
7:     senão
8:         π' ← MultipleSwap(1, 1, π)
9:     fim se
10:    retorna π'
11: fim função
```

Fonte: Kramer, 2015 [40]

4.7 Critério de aceitação

O critério de aceitação é responsável por decidir a partir de qual solução a busca deverá continuar. Na heurística FILS, tal critério corresponde a qual das soluções, entre a melhor solução da iteração atual e a melhor solução conhecida até o momento, possui a menor soma dos tempos de conclusão, C_{sum}.

4.8 Algoritmo FILS

Com base nisso, o objetivo é utilizar de problemas já apresentados na literatura para validação da solução proposta utilizando como base a estrutura do ILS considerando problemas do tipo $F \mid\mid \circ \mid\mid C_{sum}$. Portanto, o algoritmo resultante foi nomeado de FILS e sua estrutura é apresentada no Algoritmo 5.

Algoritmo 5 FILS

```
1: função Fils(t_max)
2:     t_start ← t                                  ▷ t: tempo atual
3:     π_0 ← FF(n/m)                                ▷ Cf. Algoritmo 2
4:     π ← RVND(N, π_0)                             ▷ Cf. Algoritmo 3
5:     enquanto t − t_start < t_max faça
6:         π' ← FILSPERTUBATION(π)                  ▷ Cf. Algoritmo 4
7:         π'' ← RVND(N, π')
8:         se C_sum(π'') < C_sum(π) então
9:             π ← π''
10:        fim se
11:    fim enquanto
12:    retorna π
13: fim função
```

Fonte: Autor

5 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Neste capítulo, serão apresentados os resultados obtidos através da aplicação do algoritmo variante do ILS, intitulado FILS. O algoritmo foi implementado utilizando a linguagem de programação Python com a biblioteca numpy e os experimentos realizados foram executados em um *Droplet* da *DigitalOcean* com as seguintes configurações:

- OS: Ubuntu 20.04 (LTS) x64;
- Processadores: 4 vCPUs, dos modelos Intel Xeon Broadwell (2.6 GHz) e Intel Xeon Skylake (2.7 GHz, 3.7 GHz turbo);
- RAM: 8 GB;

Os algoritmos utilizados para comparação de resultados foram o VNS4 [19], AGA e o V4AGA [96]. As instâncias serão as de Taillard [86].

Para limite de tempo de execução, os autores do VNS4 utilizaram $0.40 \times n \times m$ segundos. O valor de 0.40 representa p que, neste trabalho, foi considerado em 4 casos, sendo eles:

- $0.03 \times n \times m =$ FILS 30
- $0.06 \times n \times m =$ FILS 60
- $0.09 \times n \times m =$ FILS 90
- $0.40 \times n \times m =$ FILS 400

Para a geração da solução inicial, permanecemos com os parâmetros sugeridos por Fernandez-Viagas e Framinan [26] para o FF(n/m), com $a = 4$ e $b = 1$. Para as estruturas de vizinhança utilizadas na busca local, utilizamos as variações de l-*block insertion* com $l \in \{1, 2\}$ e de *block swap* com $(l, l') \in \{(1,1), (1,2), (1,3), (2,2), (2,3), (2,4), (3,3), (3,4), (4,4)\}$ [40]. Por fim, para o passo de perturbação, utilizamos $l = 2$ e $l' = 3$ para o MultipleSwap(l, l').

Os algoritmos foram executados 10 vezes cada conjunto de instâncias a fim de validar qual seria a melhor solução. Portanto, as tabelas a seguir mostram os dados gerados através dos respectivos testes, considerando as instâncias de Taillard de 50, 100, 200 e 500 *jobs*, iniciando em TA031, assim como os autores do VNS4 utilizaram. Serão destacados os melhores valores para mínimo e média, a fim de destacar os melhores resultados obtidos durante o experimento.

Tabela 4: Resultados dos algoritmos FILS para as instâncias com 50 *jobs* com a respectiva execução enumerada de 1 à 10. São apresentados os valores mínimo (Min), média (Med) e máximo (Max), com valores em negrito para os melhores resultados.

Instâncias	FILS 30			FILS 60			FILS 90			FILS 400		
	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
50 x 05 1	64112	67582.7	71074	64022	67432.7	70756	63884	67381.6	70646	**63699**	**67019**	69982
50 x 05 2	64478	67556.2	70886	63967	67423.1	70711	64002	67284.6	70343	**63625**	**67107.8**	70183
50 x 05 3	64265	67574.6	70965	63950	67357.8	70579	63730	67299.4	70697	**63812**	**67086**	69955
50 x 05 4	64402	67621.5	71085	64062	67382.4	70839	64006	67290.9	70474	**63635**	**67042.4**	70158
50 x 05 5	64401	67617.8	71280	64038	67443.9	70779	63787	67180.3	70407	**63464**	**67010.8**	70174
50 x 05 6	64097	67575.6	71130	64108	67399.1	70705	63825	67339.8	70606	**63649**	**66982.2**	69933
50 x 05 7	64233	67601.7	70973	63970	67446.2	70814	64073	67316.2	70578	**63626**	**67046.3**	69835
50 x 05 8	64375	67592.3	70981	64057	67427.9	70825	63792	67280.1	70582	**63740**	**67053.8**	70042
50 x 05 9	64243	67594.8	70985	63835	67360.2	70638	63885	67265.6	70640	**63794**	**67134.9**	70183
50 x 05 10	64254	67596.1	70915	63904	67378.9	70662	64081	67310.2	70529	**63683**	**67069.3**	70153
50 x 10 1	82361	88672.9	91168	82231	88347.6	90898	81950	88290.5	90899	**81237**	**87510.7**	90002
50 x 10 2	82282	88754.6	91155	82021	88467.4	90920	82138	88092.7	90529	**81655**	**87609.5**	89912
50 x 10 3	82223	88848	91328	82232	88501.7	91028	82198	88407.1	90715	**81805**	**87695.3**	90822
50 x 10 4	82364	88698.5	91120	82162	88467	91084	82104	88180.7	90702	**82159**	**87786**	90737
50 x 10 5	82415	88751.8	91499	82151	88320.9	90764	82115	87995.2	90562	**81542**	**87894.7**	90718
50 x 10 6	82370	88832.8	91407	82215	88294.2	91031	82122	88281.4	90818	**81906**	**87742.4**	90255
50 x 10 7	82534	88754.4	91007	82088	88574.4	91000	82258	88241.8	90499	**81194**	**87737.1**	90557
50 x 10 8	82420	88783.9	91388	82403	88501.5	91010	82122	88299.5	90657	**81460**	**87521.5**	90410
50 x 10 9	82382	88908.3	91405	82250	88355.7	91081	82172	88170.6	90760	**82131**	**87688**	90330
50 x 10 10	82438	88734.9	91386	82286	88476	91151	**81752**	88092.2	90607	82122	**87512.1**	90527
50 x 20 1	121026	126899.8	131821	120509	125662.2	130658	120109	125290.3	131048	**119923**	**123918.8**	128575
50 x 20 2	120734	126707.1	131516	120411	125706.1	130252	119945	125322.6	130166	**119747**	**124226.6**	128594
50 x 20 3	120989	126812.5	132089	120756	125839.7	130495	120351	125286.2	129985	**119130**	**123686.4**	127841
50 x 20 4	120974	126811.8	131354	120943	126058.4	130597	119934	125229.1	129369	**119528**	**123937.4**	127887
50 x 20 5	120818	126623.4	131372	120406	126253.7	131405	**119864**	124898.5	129404	120049	**124304**	128055
50 x 20 6	121136	126815.3	130849	120436	125867.5	129915	120252	125604.2	129972	**119408**	**124126.5**	128572
50 x 20 7	121372	126538.9	131501	120469	125928.1	129342	119938	125580.2	129640	**118915**	**124047**	127940
50 x 20 8	121243	126725	131901	120507	126568.6	131571	120226	125113.5	128644	**119424**	**124096.4**	127937
50 x 20 9	120962	126840	131926	120736	126031.1	131044	120134	125332.1	129584	**119650**	**124114.3**	127745
50 x 20 10	120755	126534.9	131928	120719	125933.8	131233	120000	125449.8	129183	**119660**	**124095.3**	127889

Tabela 5: Resultados dos algoritmos FILS para as instâncias com 100 *jobs* com a respectiva execução enumerada de 1 à 10. São apresentados os valores mínimo (Min), média (Med) e máximo (Max), com valores em negrito para os melhores resultados.

Instâncias	FILS 30			FILS 60			FILS 90			FILS 400		
	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
100 x 05 1	231670	243869.2	257869	231397	243641.2	257538	231476	243577.8	257403	**230483**	**242525**	256118
100 x 05 2	231689	243840.4	258015	231589	243622.5	257578	231503	243575.7	257504	**231078**	**242869.7**	256435
100 x 05 3	231475	243795.8	257737	231682	243643.9	257598	231013	243433	257140	**230328**	**242781**	256481
100 x 05 4	231660	243904.2	257975	231501	243629.3	257607	231457	243588.2	257297	**230449**	**242892.6**	256778
100 x 05 5	231623	243866.9	257958	231473	243636.5	257758	231206	243328.9	257407	**230675**	**242782.6**	256518
100 x 05 6	231623	243820.9	257867	231498	243624.5	257767	231212	243531.4	257447	**230551**	**242764.7**	256881
100 x 05 7	231655	243873.7	257977	231486	243722.3	257576	231396	243679	257428	**230804**	**242753.7**	256776
100 x 05 8	231699	243909.7	257876	231424	243818	257957	231246	243554.2	257448	**230819**	**242859.5**	256101
100 x 05 9	231699	243907.9	258006	231254	243636.6	257748	231373	243576.4	257816	**230662**	**242843.1**	256782
100 x 05 10	231670	243855.9	257757	231436	243713.8	257937	231527	243709.2	257533	**230877**	**242841.3**	256739
100 x 10 1	277754	299138.3	316631	277367	298252.2	314881	277530	297839.1	314296	**277023**	**296049.1**	310370
100 x 10 2	277909	299075.1	316704	277947	298337.8	314976	277181	297722.9	314327	**276711**	**296166.4**	311648
100 x 10 3	277939	298919	315630	277410	298374.2	314719	277348	297702.9	313497	**276756**	**296117.8**	310179
100 x 10 4	278114	299084.6	315740	277690	298444.9	315229	277578	297766.5	313884	**276938**	**296142.7**	311230
100 x 10 5	278221	299085.3	316436	277691	298437.1	315233	277143	297419.5	312927	**276831**	**295944.2**	310730
100 x 10 6	277890	298889.2	315851	277758	298318.6	314588	277511	297874.4	313795	**276545**	**296114.7**	311732
100 x 10 7	277975	299162.9	316060	277721	298336.2	314726	277401	298033.6	313742	**276840**	**295910.4**	311027
100 x 10 8	278151	298783.4	316478	277562	298188.5	314669	277539	297839.9	314297	**276528**	**296234.5**	311280
100 x 10 9	278270	298865.5	315954	277949	298238.9	314581	277773	297662.6	314102	**276370**	**295938.8**	311772
100 x 10 10	278074	299092.9	316193	277613	298533.9	314628	277742	297910.7	314366	**276548**	**296000.2**	310473
100 x 20 1	387593	396021.1	403741	385400	394371.4	403674	385781	393725.6	403200	**381874**	**388852.2**	399289
100 x 20 2	387550	395309.9	403796	385911	394134.5	403959	385172	392935.6	401960	**382339**	**388110**	394610
100 x 20 3	386854	395596	404679	385018	393934.6	402394	385202	393876.4	403617	**382646**	**389074.8**	396234
100 x 20 4	386336	395057.7	403483	386310	393669.9	401579	386075	393766.7	403246	**382512**	**388822.2**	395097
100 x 20 5	387457	394996.3	402001	386697	394630.2	404319	383111	391896.5	400752	**381730**	**389621.2**	398370
100 x 20 6	387270	395075.1	403308	386630	394641	401766	385120	393385.6	402003	**381705**	**388768.1**	394246
100 x 20 7	387654	395378.6	404494	385953	394440.6	404194	385442	393834.3	401878	**381868**	**389184.8**	396515
100 x 20 8	387689	395600.8	404679	386423	394455.8	402874	384897	393364.9	402559	**380951**	**388066.9**	393991
100 x 20 9	386854	395120.7	404502	386934	394516	402610	384789	393268.3	401192	**381441**	**389375.5**	396129
100 x 20 10	386336	395108	404754	386051	394521.5	402657	385873	393688.6	402427	**381176**	**388714.3**	398875

Tabela 6: Resultados dos algoritmos FILS para as instâncias com 200 e 500 *jobs* com a respectiva execução enumerada de 1 à 10. São apresentados os valores mínimo (Min), média (Med) e máximo (Max), com valores em negrito para os melhores resultados.

Instâncias	FILS 30			FILS 60			FILS 90			FILS 400		
	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
200 x 10 1	1028389	1059514	1084340	1028138	1058431.3	1080295	1026712	1057361.2	1080527	**1023330**	**1053568.6**	1075041
200 x 10 2	1028389	1059472.1	1084169	1027330	1058219.9	1080295	1026754	1056761.8	1080272	**1024198**	**1052966.8**	1071973
200 x 10 3	1028389	1059483.7	1084994	1027807	1058827.5	1082001	1026657	1057582.9	1079606	**1025961**	**1055105.2**	1078401
200 x 10 4	1028389	1059640.3	1085771	1027807	1058473.6	1081639	1026400	1057361	1079721	**1025295**	**1054371.8**	1076466
200 x 10 5	1028389	1059423.7	1084467	1028389	1058889.4	1081639	1026827	1057332.5	1080933	**1025512**	**1054875.7**	1077161
200 x 10 6	1028328	1059448.7	1084069	1028378	1058802.5	1082001	1027156	1057563.2	1080505	**1025101**	**1054380.3**	1076304
200 x 10 7	1028389	1059542.8	1085909	1028138	1058626.5	1082001	1026662	1057435.2	1079721	**1025675**	**1054832.1**	1076841
200 x 10 8	1028389	1059507.4	1084994	1028389	1058669.8	1082961	1026968	1057377.8	1079925	**1025338**	**1054704**	1075889
200 x 10 9	1028389	1059402.6	1082961	1028035	1058768.6	1081782	1026662	1057714.1	1081782	**1024521**	**1054401.1**	1078111
200 x 10 10	1028389	1059486.2	1084553	1028389	1058852.5	1082252	1027273	1057392.5	1079721	**1024418**	**1054239.6**	1077911
200 x 20 1	1272066	1290572.8	1310127	1269896	1289320.7	1314253	1268307	1288280.3	1310217	**1264583**	**1280414.8**	1304296
200 x 20 2	1270337	1290149.3	1310127	1271054	1289352	1314253	1270159	1285839.7	1308907	**1265952**	**1282192.8**	1306901
200 x 20 3	1271054	1290666.4	1311881	1271922	1288672.3	1309540	1266097	1286288	1308960	**1266461**	**1282913.2**	1307986
200 x 20 4	1271488	1290715.1	1312731	1271754	1289031.8	1309540	1270952	1288374.5	1310127	**1262434**	**1281930.1**	1303192
200 x 20 5	1272066	1290383.7	1309838	1271579	1288742.7	1310127	1268153	1286314.7	1308879	**1267208**	**1283699.7**	1305309
200 x 20 6	1269039	1290982.6	1314253	1270736	1289442.8	1313228	1270952	1288917.4	1311269	**1267368**	**1283574.9**	1305182
200 x 20 7	1269039	1290931.6	1314253	1270337	1289036.9	1312731	1271191	1288548	1310217	**1266346**	**1282293.9**	1307757
200 x 20 8	1271579	1290920.6	1311981	1270952	1289057	1309540	1269914	1288132.6	1309838	**1261741**	**1280263.6**	1300935
200 x 20 9	1269039	1290766.6	1311290	1271579	1288703.4	1309838	1270718	1288282.7	1311290	**1264844**	**1281985.1**	1303874
200 x 20 10	1272066	1290277.8	1309540	1270736	1289137.6	1311530	1270952	1288139.8	1310127	**1261717**	**1279316**	1301607
500 x 20 1	6770710	6844848.8	6921363	6770710	6842632.1	6917121	6769531	6843454.1	6918801	**6765712**	**6840158.9**	6912794
500 x 20 2	6770710	6844848.8	6921363	6770710	6843231.3	6919522	6770261	6842637.6	6916710	**6766098**	**6839753.5**	6910869
500 x 20 3	6770710	6844848.8	6921363	6770710	6843981.8	6919252	6770710	6843860.2	6919252	**6768217**	**6839952.4**	6915284
500 x 20 4	6770710	6844848.8	6921363	6770710	6844793.6	6921094	6770710	6843261.4	6917121	**6765712**	**6842728.8**	6919252
500 x 20 5	6770710	6844848.8	6921363	6770710	6844016.5	6917121	6770710	6843908.4	6918801	**6769328**	**6839887.8**	6910958
500 x 20 6	6770710	6844848.8	6921363	6770710	6842702	6916030	6770710	6843967.4	6918801	**6762933**	**6840378.8**	6913876
500 x 20 7	6770710	6844848.8	6921363	6770710	6844483.5	6918801	6770710	6844669.1	6921094	**6769943**	**6841819.1**	6917922
500 x 20 8	6770710	6844848.8	6921363	6770710	6844524.1	6918801	6770710	6842865.5	6918261	**6768449**	**6838437.1**	6917121
500 x 20 9	6770710	6844848.8	6921363	6770710	6844545.7	6918413	6770710	6844482.1	6918557	**6766533**	**6837755.5**	6911829
500 x 20 10	6770710	6844848.8	6921363	6770710	6843130	6916837	6770710	6842167.4	6912947	**6766575**	**6836910.9**	6906909

Na Tabela 4, por duas vezes o algoritmo FILS 90 obteve resultado de mínimo melhor do que o FILS 400 que, por sua vez, obteve os melhores resultados de mínimo (Min) na maioria das instâncias e o melhor valor de média (Med) absoluto.

Nas Tabelas 5 e 6, fica evidente a superioridade da solução FILS 400 sobre os demais algoritmos baseado em melhores valores de mínimo (Min) e de média (Med). Portanto, o FILS 400 será utilizado como algoritmo principal recebendo o nome de FILS para comparação com os resultados apresentados pelo VNS4, AGA e V4AGA.

Os valores para os melhores resultados (BKS) são os encontrados no artigo do VNS4, estes utilizados para comparação direta com a respectiva meta-heurística. Os autores não consideraram as instâncias abaixo de TA031, portanto, utilizaremos as instâncias a partir deste ponto para comparação, dividindo os resultados das instâncias em grupos de 50, 100, 200 e 500 *jobs*.

Serão utilizados para comparações das soluções os melhores resultados apresentados anteriormente, associados aos resultados obtidos pelos autores do VNS4, chamados de BKS, o *Mínimo* com o menor valor, a *Média* com o valor médio e o *Máximo* com o resultado máximo resultantes das execuções das instâncias.

São demonstrados nas Tabelas 7, 8 e 9 os resultados obtidos através do comparativo entre os algoritmos, destacando os novos valores de mínimo pelo símbolo ($\star$), e os melhores valores entre os algoritmos, em negrito.

Na Tabela 7, é possível observar que em instâncias com n = 50 *jobs*, o FILS é superior em 24 execuções com o melhor mínimo, sendo considerado o novo BKS, e possui os melhores valores de média em 8 instâncias. O VNS4 empatou com o V4AGA em 1 instância, porém obteve os melhores valores de média em 13 instâncias. O AGA possui o melhor valor de mínimo por 3 vezes e por 1 vez o melhor valor de média, e o V4AGA por 4 vezes é o melhor mínimo, porém por 19 vezes possui o melhor valor de média.

Na Tabela 8, é possível observar que em instâncias com n = 100 *jobs*, o FILS é superior em 19 execuções com o melhor mínimo, sendo considerado o novo BKS, e possui os melhores valores de média em 7 instâncias. O VNS4 possui os melhores valores de mínimo em 15 instâncias, e obteve os melhores valores de média em 17 instâncias. O AGA possui o melhor valor de mínimo por 2 vezes entre os algoritmos executados, desconsiderando seu valor anterior já considerado BKS por 14 vezes, e não obteve o melhor valor de média, e o V4AGA não possui melhores valores.

Na Tabela 9, é possível observar que em instâncias com n = 200 *jobs*, o FILS é superior em 9 execuções com o melhor mínimo, sendo considerado o novo BKS, e possui os melhores valores de média em 2 instâncias. O VNS4 possui os melhores valores de mínimo em 1 instância, e obteve os melhores valores de média em 7 instâncias. O AGA possui o melhor valor de média em 1 instância, e o V4AGA não possui melhores valores. Para as

instâncias com n = 500 *jobs*, o FILS possui o melhor valor de mínimo em 2 instâncias, já o VNS4 possui os melhores valores de mínimo em 8 instâncias e apresenta os melhores valores de média em todas as instâncias. AGA e V4AGA não possuem valores competitivos apresentados, desconsiderando os valores já apresentados de BKS.

Tabela 7: Resultados dos algoritmos VNS4, AGA, V4AGA e FILS para as instâncias com 50 *jobs* com a respectiva execução enumerada de 1 à 10. São apresentados os melhores valores encontrados (BKS), mínimo (Min), média (Med) e máximo (Max), com valores em negrito para os melhores resultados entre os algoritmos e (⋆) para o melhor valor de mínimo.

Instâncias	BKS	Algoritmo	VNS 4			AGA			V4AGA			FILS		
			Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
50 x 05 1	64803	VNS4 / V4AGA	64803	64917,17	65083	**64817**	64891,4	64982	64803	**64857,47**	64922	**63699**⋆	67019	69982
50 x 05 2	68051	hDDE	68083	68203,1	68396	68095	68174,5	68262	68059	68131,5	68215	**63625**⋆	**67107,8**	70183
50 x 05 3	63162	hDDE	**63195**	63384,4	63563	63241	**63420,07**	63627	**63195**	63335,47	63540	63812	67086	69955
50 x 05 4	68241	HGLS	68241	68535,43	68730	68241	68441,37	68627	68241	68408,17	68566	**63635**⋆	**67042,4**	70158
50 x 05 5	69360	hDDE	69392	69561,2	69699	69466	69554,27	69639	69414	69517,47	69632	**63464**⋆	**67010,8**	70174
50 x 05 6	66841	hDDE	66865	67056,77	67263	66961	67020,67	67091	66841	**66958,27**	67100	**63649**⋆	66982,2	69933
50 x 05 7	66253	AGA	66273	66427,77	66635	66271	66356,37	66445	66261	**66311,27**	66415	**63626**⋆	67046,3	69835
50 x 05 8	64359	AGA / V4AGA	64381	64591,5	64845	64359	64472,13	64663	64359	**64426,4**	64536	**63740**⋆	67053,8	70042
50 x 05 9	62981	AGA / V4AGA	63062	63145,63	63224	**62981**	63104,43	63202	**62981**	**63086,4**	63172	63794	67134,9	70183
50 x 05 10	68811	HGLS	68884	69137,87	69394	68929	69097,27	69267	68951	69072,37	69174	**63683**⋆	**67069,3**	70153
50 x 10 1	87143	hDDE	88100	88215,2	88313	87225	87683,2	87976	87252	**87584,43**	87779	**81237**⋆	87510,7	90002
50 x 10 2	82820	HGLS	83042	83270,27	83681	83059	83249	83631	82954	**83239,27**	83523	**81655**⋆	87609,5	89912
50 x 10 3	79987	HGLS	**80069**	80328,17	80657	80092	80363,73	80570	80080	**80211,8**	80432	81805	87695,3	90822
50 x 10 4	86466	HGLS	86600	86915,7	87181	86534	86883,1	87117	86579	**86828,6**	87028	**82159**⋆	87786	90737
50 x 10 5	86391	HGLS	86655	86861,6	87183	86623	86864,17	87157	86579	**86770,13**	86943	**81542**⋆	87894,7	90718
50 x 10 6	86643	V4AGA	86840	87027,4	87412	86740	86970,6	87375	86643	**86831,2**	87156	**81906**⋆	87742,4	90255
50 x 10 7	88778	AGA	89020	89409,67	89809	88778	89293,9	89629	89009	89260,57	89468	**81194**⋆	**87737,1**	90557
50 x 10 8	86839	HGLS	86848	**87138,33**	87503	86926	87171	87658	86940	87199,73	87481	**81460**⋆	87521,5	90410
50 x 10 9	85548	HGLS	85555	86015,67	86355	85659	**86008,4**	86214	85679	86018,03	86276	**82131**⋆	87688	90330
50 x 10 10	87998	DABC	88214	88529,4	89020	87998	88388,53	88817	88169	88419,67	88715	**82122**⋆	**87512,1**	90527
50 x 20 1	125831	VNS	125852	126401	127008	125831	126379,83	126796	125831	126169,67	126697	**119923**⋆	**123918,8**	128575
50 x 20 2	119247	V4AGA	**119270**	119637,8	120253	**119274**	119538,6	120060	119247	**119438,57**	119681	119747	124226,6	128594
50 x 20 3	116459	HGLS	116792	117177,93	117766	116645	117070,77	117526	**116536**	**116959,63**	117289	119130	123686,4	127841
50 x 20 4	120766	HGLS	120972	121506,13	122081	120766	121193,93	121739	120770	**120986**	121288	**119528**⋆	123937,4	127887
50 x 20 5	118405	V4AGA	118391	119090,23	119575	118460	118850,6	119204	**118391**	**118718,57**	119104	120049	124304	128055
50 x 20 6	120703	hDDE	120792	121283,33	121634	120791	121167,7	121637	120758	**121028,03**	121500	**119408**⋆	124126,5	128572
50 x 20 7	123018	HGLS	123237	123834,53	124542	123058	123648	124062	123068	**123406,27**	123801	**118915**⋆	124047	127940
50 x 20 8	122520	HGLS	122708	123245,87	123772	122703	123099,73	123504	122532	**122949**	123350	**119424**⋆	124096,4	127937
50 x 20 9	121872	HGLS	121872	122450,57	123170	122073	122348,37	122853	121872	**122220,2**	122610	**119650**⋆	124114,3	127745
50 x 20 10	124079	hDDE	124182	124712	125463	124361	124720,7	125116	124225	124552,73	124912	**119660**⋆	**124095,3**	127889

Tabela 8: Resultados dos algoritmos VNS4, AGA, V4AGA e FILS para as instâncias com 100 *jobs* e execuções enumeradas de 1 à 10. São apresentados os melhores valores encontrados (BKS), mínimo (Min), média (Med) e máximo (Max), com valores em negrito para os melhores resultados entre os algoritmos e (⋆) para o melhor valor de mínimo.

Instâncias	BKS	Algoritmo	VNS 4			AGA			V4AGA			FILS		
			Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
100 x 05 1	253926	AGA	254271	254706,53	255098	254540	255160,57	256202	255153	255572,2	255932	**230483⋆**	**242525**	256118
100 x 05 2	242886	AGA	243170	243739,1	244155	243452	244344,97	244970	244106	244796,97	245164	**231078⋆**	**242869,7**	256435
100 x 05 3	238280	AGA	**238452**	238802,57	239161	239223	239498,87	239982	239341	239777,87	240105	**230328⋆**	242781	256481
100 x 05 4	228169	AGA	**228494**	228845,23	229248	**228321**	228900,8	229323	228995	229551,3	229917	230449	242892,6	256778
100 x 05 5	240810	AGA	**241008**	241557,3	242156	241261	241879,4	242407	242097	242393,73	242663	**230675⋆**	242782,6	256518
100 x 05 6	232876	AGA	**233277**	233771,77	234203	233655	234287,63	234832	234389	234793,77	235099	**230551⋆**	242764,7	256881
100 x 05 7	240874	VNS4	**240874**	241449,47	241873	241402	241944,13	242556	241807	242120,83	242570	**230804⋆**	242753,7	256776
100 x 05 8	231716	hDDE	**231782**	232267,3	232962	232224	232850,9	233517	232578	233157,57	233449	**230819⋆**	242859,5	256101
100 x 05 9	248679	AGA	248652	249283,93	249885	249455	250167,23	251145	250032	250386,8	250814	**230662⋆**	**242843,1**	256782
100 x 05 10	243518	AGA	243822	244396,67	245120	243933	244493,73	244963	244386	244890	245224	**230877⋆**	**242841,3**	256739
100 x 10 1	299999	VNS4	299999	301055,93	302238	301032	302020,8	302941	302014	302761,1	303342	**277023⋆**	**296049,1**	310370
100 x 10 2	275298	AGA	**275989**	**276894,57**	278107	276270	277873,57	279732	277664	278560,2	279296	276711	296166,4	311648
100 x 10 3	288707	AGA	289505	**290715,23**	292016	290021	291220,03	292411	290687	291935,37	292652	**276756⋆**	296117,8	310179
100 x 10 4	302635	AGA	303502	305109,3	306632	303583	305185,2	306365	304301	305459,6	306206	**276938⋆**	**296142,7**	311230
100 x 10 5	285643	AGA	285692	**286403,43**	286992	286184	287938,87	288780	288115	288620,47	289050	**276831⋆**	295944,2	310730
100 x 10 6	271475	AGA	**271822**	**272750,17**	273594	272664	273816,37	274845	273294	274349,37	274973	276545	296114,7	311732
100 x 10 7	280921	hDDE	281312	**282428,2**	283607	281496	282708,73	283968	282082	283432,13	284233	**276840⋆**	295910,4	311027
100 x 10 8	292471	AGA	292636	**294095,27**	295093	292883	294923,17	296263	294566	295386,53	295963	**276528⋆**	296234,5	311280
100 x 10 9	303594	VNS4	303594	305259,17	307273	304500	305558,27	306757	305489	306146,8	306890	**276370⋆**	**295938,8**	311772
100 x 10 10	293053	VNS4	293053	**293624,73**	294517	293131	294539,33	295471	295392	296268,03	297161	**276548⋆**	296000,2	310473
100 x 20 1	368262	VNS4	**368262**	**370046,03**	371804	368598	371452,33	373315	372010	372622,2	373363	381874	388852,2	399289
100 x 20 2	373335	AGA	374723	**376116,3**	377507	**373335**	377392,3	378755	376929	378711,57	379632	382339	388110	394610
100 x 20 3	372057	hDDE	**372837**	**374393,87**	376397	373847	375444,73	376829	374684	376034,27	376932	382646	389074,8	396234
100 x 20 4	374205	DABC	**375507**	**377226,9**	378564	376192	377947,5	380017	377819	379621,23	380950	382512	388822,2	395097
100 x 20 5	370646	hDDE	**371103**	**372902,9**	374756	371960	374128,13	376779	374044	374984,07	376037	381730	389621,2	398370
100 x 20 6	373689	DABC	**375059**	**376932,07**	379932	375410	377131,97	378480	376335	378033,1	379355	381705	388768,1	394246
100 x 20 7	375188	DABC	**375530**	**377536,27**	379501	376622	378723,2	380847	378661	379948,67	380980	381868	389184,8	396515
100 x 20 8	386147	VNS4	386147	**388428,87**	390065	388188	389809,8	391039	389099	390638,33	391393	**380951⋆**	388066,9	393991
100 x 20 9	376635	VNS4	**376635**	**378624,4**	379864	377506	379892,9	381867	379735	380934,53	381890	381441	389375,5	396129
100 x 20 10	380725	DABC	381850	**383113,17**	384542	381640	383924,33	385650	383704	385025,6	386109	**381176⋆**	388714,3	398875

Tabela 9: Resultados dos algoritmos VNS4, AGA, V4AGA e FILS para as instâncias com 200 e 500 *jobs* e execuções enumeradas de 1 à 10. São apresentados os melhores valores encontrados (BKS), mínimo (Min), média (Med) e máximo (Max), com valores em negrito para os melhores resultados entre os algoritmos e (⋆) para o melhor valor de mínimo.

Instâncias	BKS	Algoritmo	VNS 4			AGA			V4AGA			FILS		
			Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
200 x 10 1	1049830	AGA	1054656	1057543,6	1059668	1055059	1057977,37	1062645	1056691	1058526,27	1060164	**1023330⋆**	**1053568,6**	1075041
200 x 10 2	1036427	AGA	1041239	**1043194,37**	1044955	1046307	1049223,63	1051620	1047212	1050751,73	1052940	**1024198⋆**	1052966,8	1071973
200 x 10 3	1048561	VNS4	1048561	**1051289,33**	1052709	1055058	1058518,8	1061652	1055560	1059119,9	1061595	**1025961⋆**	1055105,2	1078401
200 x 10 4	1033110	AGA	1035709	**1039703,17**	1042419	1038221	1043010,2	1045352	1041728	1044950,77	1047220	**1025295⋆**	1054371,8	1076466
200 x 10 5	1037392	VNS4	1037392	**1042944,3**	1046322	1044582	1048216,2	1050798	1045074	1047949,6	1051097	**1025512⋆**	1054875,7	1077161
200 x 10 6	1010243	VNS4	**1010243**	1015037,87	1018330	1012577	**1014715,53**	1017524	1016286	1019580,43	1022676	1025101	1054380,3	1076304
200 x 10 7	1058211	VNS4	1058211	1062100,8	1064875	1061660	1065440,47	1069826	1066134	1068513	1070790	**1025675⋆**	**1054832,1**	1076841
200 x 10 8	1046284	VNS4	1046284	**1048404,33**	1049503	1053464	1055184,67	1055244	1054335	1055199,23	1055244	**1025338⋆**	1054704	1075889
200 x 10 9	1026137	AGA	1026701	**1029059,87**	1031123	1032283	1038196,93	1041745	1032086	1037533,1	1040065	**1024521⋆**	1054401,1	1078111
200 x 10 10	1035409	AGA	1035763	**1038808,6**	1041619	1040107	1044485	1048501	1042530	1045444,87	1047191	**1024418⋆**	1054239,6	1077911
200 x 20 1	1231266	VNS4	**1231266**	**1234531,57**	1237772	1239042	1246325	1255010	1241475	1244061,77	1247595	1264583	1280414,8	1304296
200 x 20 2	1249948	VNS4	**1249948**	**1255873,4**	1262029	1258086	1262431,17	1268098	1254848	1258739,2	1262450	1265952	1282192,8	1306901
200 x 20 3	1272659	VNS4	1272659	**1278632,23**	1283811	1282691	1286911,5	1291336	1279039	1281892,13	1284783	**1266461⋆**	1282913,2	1307986
200 x 20 4	1243223	AGA	**1245410**	**1252900,37**	1256618	1255058	1258748,13	1263589	1255161	1258850,37	1261803	1262434	1281930,1	1303192
200 x 20 5	1229946	VNS4	**1229946**	**1232333,27**	1235308	1236728	1243420,93	1248240	1235025	1240992,8	1245901	1267208	1283699,7	1305309
200 x 20 6	1230942	VNS4	**1230942**	**1234829,03**	1238945	1241208	1245126,1	1250903	1238380	1242331,67	1246591	1267368	1283574,9	1305182
200 x 20 7	1247268	VNS4	**1247268**	**1249825,53**	1252136	1256251	1260687,6	1264656	1254281	1258977,77	1262241	1266346	1282293,9	1307757
200 x 20 8	1248110	AGA	**1249994**	**1256256,23**	1261005	1260078	1264553,4	1270582	1256926	1259611,47	1262518	1261741	1280263,6	1300935
200 x 20 9	1233099	VNS4	**1233099**	**1238289,37**	1241732	1242994	1248213,6	1254083	1243613	1247900	1251473	1264844	1281985,1	1303874
200 x 20 10	1250596	AGA	**1255584**	**1259114,37**	1265328	1266536	1269592,03	1273434	1261324	1264965,87	1268892	1261717	1279316	1301607
500 x 20 1	6707778	VNS4	**6707778**	**6717094,47**	6720035	6730350	6741399,4	6752384	6752936	6752936	6752936	6765712	6840158,9	6912794
500 x 20 2	6827845	VNS4	6827845	**6841021,37**	6845619	6862064	6872985,77	6884756	6855359	6875904,53	6895356	**6766098⋆**	6839753,5	6910869
500 x 20 3	6730079	VNS4	**6730079**	**6736537,9**	6738500	6807611	6818359,7	6832014	6787804	6809057,87	6839387	6768217	6839952,4	6915284
500 x 20 4	6748315	VNS4	**6748315**	**6752778,73**	6754596	6810530	6831271,43	6842594	6812323	6847534,43	6870660	6765712	6842728,8	6919252
500 x 20 5	6708168	VNS4	**6708168**	**6713007,4**	6714732	6781620	6793516,23	6806130	6781815	6806261,73	6836943	6769328	6839887,8	6910958
500 x 20 6	6734810	VNS4	**6734810**	**6745499,4**	6749510	6777530	6777530	6777530	6777530	6777530	6777530	6762933	6840378,8	6913876
500 x 20 7	6691468	AGA	**6701864**	**6716096,4**	6721695	6773789	6778006,57	6778152	6756862	6772691,77	6778152	6769943	6841819,1	6917922
500 x 20 8	6777529	VNS4	6777529	**6781793,8**	6784628	6799366	6811204,33	6834007	6808222	6838811,1	6848602	**6768449⋆**	6838437,1	6917121
500 x 20 9	6714308	VNS4	**6714308**	**6726322,23**	6729076	6766456	6780304,2	6790198	6749807	6769745,17	6787560	6766533	6837755,5	6911829
500 x 20 10	6755838	VNS4	**6755838**	**6758477,83**	6759219	6821787	6836192,5	6845923	6777091	6825545,3	6842772	6766575	6836910,9	6906909

Com isso, o FILS possui 55 novos melhores valores de mínimo para 90 instâncias, tendo o maior valor de BKS gerado entre as soluções apresentadas, sendo o VNS4 em 28 instâncias, sendo 2 empates, AGA em 2 instâncias (49 considerando valores anteriormente apresentados e os citados pelos autores do VNS4), e o V4AGA em 5 instâncias, sendo 3 empatadas.

Para análise de robustez das soluções, serão apresentadas na Tabela 10 os valores de coeficiente de variação, resultando da divisão do desvio padrão pela média, sendo estes apresentados em valores médios para o conjunto de instâncias de 50, 100, 200 e 500 *jobs*.

Tabela 10: Resultados dos coeficientes de variação dos algoritmos VNS4, AGA, V4AGA e FILS para as instâncias com 50, 100, 200 e 500 *jobs* e a média dos coeficientes de variação (Coef. de variação) das respectivas execuções de 1 a 10.

Instâncias	VNS 4	AGA	V4AGA	FILS
	Coef. de variação	Coef. de variação	Coef. de variação	Coef. de variação
50 x 05	0.15%	0.10%	0.09%	3.38%
50 x 10	0.12	0.18%	0.15%	2.87%
50 x 20	0.09%	0.17%	0.14%	2.13%
100 x 05	0.04%	0.13%	0.09%	3.07%
100 x 10	0.04%	0.22%	0.13%	3.72%
100 x 20	0.03%	0.24%	0.14%	1.20%
200 x 10	0.01%	0.16%	0.11%	1.40%
200 x 20	0.01%	0.19%	0.14%	1.03%
500 x 20	0.00%	0.08%	0.14%	0.70%

A solução FILS, por apresentar resultados para o coeficiente de variação abaixo de 3.72%, demonstra que a solução é robusta e com baixa variabilidade de resultados mas, de acordo com a Tabela 10, comparado a outras soluções, a heurística FILS possui maior variabilidade. Tal característica aparenta ser boa para alcançar novas regiões em instâncias pequenas, mas mostrou-se ineficaz para melhorar instâncias grandes. Devido ao aumento considerável da região de exploração, tal variação deve se perder neste maior espaço de busca. Vale ressaltar a interessante característica de decrescimento contínuo a partir das instâncias de 200 $jobs \times 10$ máquinas abaixo de 1.40%.

6 CONCLUSÕES

Este trabalho propôs uma heurística para resolução de problemas do tipo $F \mid\mid \circ \mid\mid C_{sum}$. Com base nisso, a solução desenvolvida é testada em problemas já apresentados na literatura para validação da mesma, utilizando como base a estrutura da meta-heurística ILS [52], utilizando o algoritmo FF [26] para geração da solução inicial, RVND [55] para o procedimento de busca local, estruturas de vizinhança para refinamento baseadas em *block insertion* [40] e *block swap* [40], e uma estrutura de perturbação de solução corrente *FILSPERTUBATION* [40], gerando assim o algoritmo nomeado de FILS.

O FILS foi testado utilizando as instâncias de Taillard [86] para problemas do tipo *Flow Shop*, com instâncias 50 e 100 *jobs* para 5, 10 e 20 máquinas, 200 *jobs* para 10 e 20 máquinas, e 500 *jobs* para 20 máquinas. Foram utilizados para calibração do algoritmo, testes envolvendo tempo de execução de 30, 60, 90 e 400 milissegundos, onde concluiu-se que a execução utilizando 400 milissegundos foi superior aos demais tempos, sendo este adotado como solução principal.

Foram realizados os mesmos testes com outros algoritmos presentes na literatura para o mesmo tipo de problema, sendo os principais o VNS4 [19], AGA e V4AGA [96]. Foram utilizados para comparações das soluções os melhores resultados citados pelos autores do VNS4, chamados de BKS, o *Min* com o menor valor, a *Med* com o valor médio e o *Max* com o resultado máximo resultantes das execuções das instâncias. o FILS resultou em 55 novos melhores valores de mínimo para 90 instâncias, tendo o maior valor de BKS gerado entre as soluções apresentadas, sendo o VNS4 em 28 instâncias, com 2 empates, AGA em 49 instâncias, considerando valores anteriormente apresentados e os citados pelos autores do VNS4, e o V4AGA em 5 instâncias, sendo 3 empatadas.

Para análise de robustez da solução, foram utilizados os valores de coeficiente de variação médio para cada grupo de instâncias, onde o FILS apresentou valores abaixo de 4%, com uma queda contínua a partir das instâncias de 100 *jobs* x 10 máquinas, sendo de 3.72% e decaindo bruscamente para 1.20% nas instâncias de 100 *jobs* x 20 máquinas, até chegar à 0.7% nas instâncias de 500 *jobs* x 20 máquinas, sendo assim uma solução robusta, porém apresentando amplitude de variações maiores que os demais algoritmos utilizados para comparação, os quais apresentam resultados abaixo de 1% desde as primeiras instâncias.

Apesar do algoritmo ser mais eficiente do que o VNS, hDDE, DABC, HGLS, VNS4, AGA e V4AGA de acordo com os resultados apresentados pelos autores do VNS4, para as instâncias de Taillard, seria interessante a implementação de uma estrutura de vizinhança com um modelo matemático. Outro fator importante seria a análise das estruturas de vizinhança que estão provocando a amplitude de variação dos resultados acima dos demais algoritmos apresentados, porém garantindo melhores resultados de média em 17 instâncias, sendo estas sugestões para trabalhos futuros.

REFERÊNCIAS

[1] AKYOL, D.; BAYHAN, G. Multi-machine earliness and tardiness scheduling problem: an interconnected neural network approach. *The International Journal of Advanced Manufacturing Technology*, v. 37, n. 5-6, p. 576–588, 2008.

[2] ALVES, Rui; DELGADO, Catarina. Programação Linear Inteira. Faculdade de Economia Universidade Porto. Setembro, 1997. Disponível em: <https://repositorio-aberto.up.pt/bitstream/10216/74369/2/40539.pdf>. Acesso em: 3 de junho de 2019.

[3] AMORIM, R. Um estudo sobre formulações matemáticas e estratégias algorítmicas para problemas de escalonamento em máquinas paralelas com penalidades de antecipação e atraso. Master's thesis, Programa de Pós-Graduação em Informática, Instituto de Computação, Universidade Federal do Amazonas, Manaus, AM, Brazil. In Portuguese, 2013.

[4] AMORIM, R.; DIAS, B.; DE FREITAS, R.; UCHOA, E. A hybrid genetic algorithm with local search approach for e/t scheduling problems on identical parallel machines. *Proceedings of the* 15*th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '13 Companion, p. 63–64, New York, NY, USA. ACM, 2013.

[5] ANGHINOLFI, D.; PAOLUCCI, M. Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. *Computers & Operations Research*, v. 34, n. 11, p. 3471–3490, 2007.

[6] ARMENTANO,V.A.;DE FRANÇA FILHO, M.F.Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach. *European Journal of Operational Research*, v. 183, n. 1, p. 100 – 114, 2007.

[7] AZIZOGLU, M.; KIRCA, O. On the minimization of total weighted flow time with identical and uniform parallel machines. *European Journal of Operational Research*, v. 113, n. 1, p. 91 – 100, 1999.

[8] AZIZOGLU, M.; KIRCA, O. Tardiness minimization on parallel machines. *International Journal of Production Economics*, v. 55, n. 2, p. 163 – 168, 1998.

[9] BALAKRISHNAN, N.; KANET, J. J.; SRIDHARAN, V. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers & Operations Research*, v. 26, n. 2, p. 127 – 141, 1999.

[10] BANK, J.; WERNER, F. Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modelling*, v. 33, n. 4 - 5, p. 363 – 383, 2001.

[11] BANSAL, Sorav P. Minimizing the Sum of Completion Times of n Jobs over m Machines in a Flowshop: A Branch and Bound Approach. A I I E Transactions, v. 9, n. 3, p. 306–311, set. 1977. ISSN 0569-5554. DOI: 10.1080/05695557708975160.

[12] BAPTISTE, P.; JOUGLET, A.; SAVOUREY, D. Lower bounds for parallel machine scheduling problems. *International Journal of Operational Research*, v. 3, n. 6, p. 643 – 664, 2008.

[13] BELOUADAH, H.; POTTS, C. Scheduling identical parallel machines to minimize total weighted completion time. *Discrete Applied Mathematics*, v. 48, n. 3, p. 201 – 218, 1994.

[14] BILGE, U.; KIRAC , F.; KURTULAN, M.; PEKGUN, P. A tabu search algorithm for parallel machine total tardiness problem. *Computers & Operations Research*, v. 31, n. 3, p. 397 – 414, 2004.

[15] BISKUP, D.; FELDMANN, M. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers & Operations Research*, v. 28, n. 8, p. 787 – 801, 2001.

[16] BISKUP, D.; HERRMANN, J.; GUPTA, J. N. Scheduling identical parallel machines to minimize total tardiness. *International Journal of Production Economics*, v. 115, n. 1, p. 134 – 142, 2008.

[17] BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, v. 35, n. 3, p. 268–308, 2003.

[18] CHEN, Z.-L.; POWELL, W. B. Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing*, v. 11, n. 1, p. 78–94, 1999.

[19] COSTA, Wagner M.; GOLDBARG, Marco C.; GOLDBARG, Elizabeth G.. New VNS heuristic for total flowtime flowshop scheduling problem. *Expert Systems with Applications*, v. 39, n. 9, p. 8149–8161, 2012.

[20] CROCE, F. D.; GARAIX, T.; GROSSO, A. Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem. *Computers & Operations Research*, v. 39, n. 6, p. 1213 – 1217. Special Issue on Scheduling in Manufacturing Systems, 2012.

[21] EOM, D.-H.; SHIN, H.-J.; KWUN, I.-H.; SHIM, J.-K.; KIM, S.-S. Scheduling jobs on parallel machines with sequence-dependent family set-up times. *The International Journal of Advanced Manufacturing Technology*, v. 19, n. 12, p. 926–932. ISSN 0268-3768, 2002.

[22] FANJUL-PEYRO, L.; PEREA, F.; RUIZ, R. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, V. 260, P. 482–493, 2017.

[23] FANJUL-PEYRO, L.; RUIZ, R. Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers & Operations Research*, v.38, p. 301-309, 2011.

[24] FARIAS, K.; KRAMER, A.; SUBRAMANIAN, A. Efficient local search limitation strategy for single machine total weighted tardiness scheduling with sequence-dependent setup times. Relatório técnico, Universidade Federal da Paraíba, Brazil. URL http://arxiv.org/abs/1501.05882, 2015.

[25] FENG, G.; LAU, H. Efficient algorithms for machine scheduling problems with earliness and tardiness penalties. *Annals of Operations Research*, v. 159, n. 1, p. 83–95, 2008.

[26] FERNANDEZ-VIAGAS, Victor;FRAMINAN, Jose M.. A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers & Operations Research*, v. 52, n. 1, p. 68-80, 2015.

[27] GOKALP, Osman; UGUR, Aybars. A multi-start ILS–RVND algorithm with adaptive solution acceptance for the CVRP. Soft Computing, v. 24, n. 4, p. 2941–2953, fev. 2020. ISSN 1433-7479. DOI: 10.1007/s00500-019-04072-6.

[28] GRAHAM, R.; LAWLER, E.; LENSTRA, J.; KAN, A. Optimization and approximation in deterministic sequencing and scheduling: a survey. P.L. HAMMER, E. J.; KORTE, B. (Eds.), Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization. Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver, volume 5 of Annals of Discrete Mathematics, p. 287 – 326. Elsevier, 1979.

[29] GUINET, A. Scheduling independent jobs on uniform parallel machines to minimize tardiness criteria. *Journal of Intelligent Manufacturing*, v. 6, n. 2, p. 95–103, 1995.

[30] HELLER, Jack. Combinatorial, probabilistic and statistical aspects of an M × J scheduling problem. New York: Courant Institute of Mathematical Sciences, New York University, 1959.

[31] HELLER, Jack. Some Numerical Experiments for an M × J Flow Shop and Its Decision-Theoretical Aspects. Operations Research, v. 8, n. 2, p. 178–184, 1960. ISSN 0030-364X.

[32] HENNESSY, Jhon; PATTERSON, David. Computer Architecture: A Quantitative Approach. *Elsevier*, 2007.

[33] IGNALL, Edward; SCHRAGE, Linus. Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems. Operations Research, v. 13, n. 3, p. 400–412, jun. 1965. ISSN 0030-364X, 1526-5463. DOI: 10.1287/opre.13.3.400.

[34] JOHNSON, S. M. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly, v. 1, n. 1, p. 61–68, mar. 1954. ISSN 00281441, 19319193. DOI: 10.1002/nav.3800010110.

[35] JOUGLET, A.; SAVOUREY, D. Dominance rules for the parallel machine total weighted tardiness scheduling problem with release dates. *Computers & Operations Research*, v. 38, n. 9, p. 1259 – 1266, 2011.

[36] JÜNGER, Michael; MALLACH, Sven. An integer programming approach to optimal basic block instruction scheduling for single-issue processors. *Discrete Optimization*, vol. 22, p. 37-65, 2015.

[37] KEDAD-SIDHOUM, S.; SOLIS, Y. R.; SOURD, F. Lower bounds for the earliness-tardiness scheduling problem on parallel machines with distinct due dates. *European Journal of Operational Research*, v. 189, n. 3, p. 1305 – 1316, 2008.

[38] KIM, S.-I.; CHOI, H.-S.; LEE, D.-H. Tabu search heuristics for parallel machine scheduling with sequence-dependent setup and ready times. GAVRILOVA, M.; GERVASI, O.; KUMAR, V.; TAN, C.; TANIAR, D.; LAGANÁ, A.; MUN, Y.; CHOO, H. (Eds.), *Computational Science and Its Applications - ICCSA* 2006, volume 3982 of *Lecture Notes in Computer Science*, p. 728–737. Springer Berlin Heidelberg, 2006.

[39] KOULAMAS, C. Decomposition and hybrid simulated annealing heuristics for the parallel-machine total tardiness problem. *Naval Research Logistics* (*NRL*), v. 44, n. 1, p. 109–125, 1997.

[40] KRAMER, Arthur Harry Frederico Ribeiro. UM MÉTODO HEURÍSTICO PARA A RESOLUÇÃO DE UMA CLASSE DE PROBLEMAS DE SEQUENCIAMENTO DA

PRODUÇÃO ENVOLVENDO PENALIDADES POR ANTECIPAÇÃO E ATRASO. 2015. Dissertação – Departamento de Engenharia de Produção, Universidade Federal da Paraíba, João Pessoa, 2015.

[41] LEE, J.-H.; YU, J.-M.; LEE, D.-H. A tabu search algorithm for unrelated parallel machine scheduling with sequence and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, v. 69, n. 9-12, p. 2081–2089, 2013.

[42] LEE, Y. H.; PINEDO, M. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, v. 100, n. 3, p. 464 – 474, 1997.

[43] LENSTRA, Jan Karel; RINNOOY KAN, Alexander Hendrik George; BRUCKER, Peter J. Complexity of Machine Scheduling Problems. In: ANNALS of Discrete Mathematics. [S.l.]: Elsevier, 1977. v. 1. P. 343–362. ISBN 9780720407655. DOI: 10.1016/S0167-5060(08)70743-X.

[44] LI, K.; LEUNG, J.-T.; CHENG, B.-Y. An agent-based intelligent algorithm for uniform machine scheduling to minimize total completion time. *Applied Soft Computing*, v. 25, n. 0, p. 277 – 284, 2014.

[45] LI, K.; LIN YANG, S. Non-identical parallel-machine scheduling research with minimizing total weighted completion times: Models, relaxations and algorithms. *Applied Mathematical Modelling*, v. 33, n. 4, p. 2145 – 2158, 2009.

[46] LIAW, C.-F.; LIN, Y.-K.; CHENG, C.-Y.; CHEN, M. Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers & Operations Research*, v. 30, n. 12, p. 1777 – 1789, 2003.

[47] LIN, SW, YING, KC; HUANG, CY. Multiprocessor task scheduling in multistage hybrid flowshops: A hybrid artificial bee colony algorithm with bi-directional planning. *Computers & Operations Research*, vol. 40(5), p. 1186–1195, 2013a.

[48] LIN, Y.-K. Fast LP models and algorithms for identical jobs on uniform parallel machines. *Applied Mathematical Modelling*, v. 37, n. 5, p. 3436 – 3448, 2013.

[49] LIN, Y.-K.; HSIEH, F.-Y. Unrelated parallel machine scheduling with setup times and ready times. *International Journal of Production Research*, v. 52, n. 4, p. 1200–1214, 2014.

[50] LIU, Jiyin; REEVES, Colin R.. Constructive and composite heuristic solutions to the P//*sum*Ci scheduling problem. *European Journal of Operational Research*, v. 132, n. 2, p. 439–452, 2001.

[51] LOGENDRAN, R.; MCDONELL, B.; SMUCKER, B. Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research*, v. 34, n. 11, p. 3420 – 3438, 2007.

[52] LOURENCO, H. R.; MARTIN, O. C.S; STÜTZLE, T; GLOVER, F.; KOCHENBERGER, G. A. (Eds.). Iterated Local Search. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, p. 320–353. Springer US, 2003.

[53] MARTINELLI, R.; POGGI, M.; SUBRAMANIAN, A. Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, v. 40, n. 8, p. 2145 – 2160, 2013.

[54] MASON, S. J.; JIN, S.; JAMPANI, J. A moving block heuristic to minimise earliness and tardiness costs on parallel machines. *International Journal of Production Research*, v. 47, n. 19, p. 5377–5390, 2009.

[55] MLADENOVIC, P.; HANSEN, P.. Variable Neighborhood Search. *Computers & Operations Research*, v. 24, n. 11, p. 1097–1100, 1997.

[56] M'HALLAH, R.; AL-KHAMIS, T. Minimising total weighted earliness and tardiness on parallel machines using a hybrid heuristic. *International Journal of Production Research*, v. 50, n. 10, p. 2639–2664, 2012.

[57] NESSAH, R.; YALAOUI, F.; CHU, C. A branch-and-bound algorithm to minimize total weighted completion time on identical parallel machines with job release dates. *Computers & Operations Research*, v. 35, n. 4, p. 1176 – 1190, 2008.

[58] NOGUEIRA,J.P.C.M.;ARROYO,J.E.C.;VILLADIEGO,H.M.M.;GONC ALVES, L. B. Hybrid GRASP heuristics to solve an unrelated parallel machine scheduling problem with earliness and tardiness penalties. *Electronic Notes in Theoretical Computer Science*, v. 302, n. 0, p. 53 – 72. Proceedings of the XXXIX Latin American Computing Conference (CLEI 2013), 2014.

[59] OMAR, M. K.; TEO, S. C. Minimizing the sum of earliness/tardiness in identical parallel machines schedule with incompatible job families: An improved MIP approach. *Applied Mathematics and Computation*, v. 181, n. 2, p. 1008 – 1017, 2006.

[60] PAN, Quan-Ke; RUIZ, Rubén. Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, v. 222, n. 1, p. 31 – 43, 2012.

[61] PEARL, Judea. Heuristics: intelligent search strategies for computer problem solving. Reading, Mass: Addison-Wesley Pub. Co, 1984. (The Addison-Wesley series in artificial intelligence). ISBN 9780201055948.

[62] PENNA, P. H. V.; SUBRAMANIAN, A.; OCHI, L. S. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, v. 19, p. 201–232, 2013.

[63] PESSOA, A.; UCHOA, E.; ARAGÃO, M. P.; RODRIGUES, R. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, v. 2, n. 3-4, p. 259–290, 2010.

[64] PFUND, M.; FOWLER, J. W.; GADKARI, A.; CHEN, Y. Scheduling jobs on parallel machines with setup times and ready times. *Computers & Industrial Engineering*, v. 54, n. 4, p. 764 – 782, 2008.

[65] POLYAKOVSKIY, S.; M'HALLAH, R. A multi-agent system for the weighted earliness tardiness parallel machine problem. *Computers & Operations Research*, v. 44, p. 115 – 136, 2014.

[66] RADHAKRISHNAN, S.; VENTURA, J. A. Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research*, v. 38, n. 10, p. 2233–2252, 2000.

[67] RAJA, K.; SELLADURAI, V.; SARAVANAN, R.; ARUMUGAM, C. Earliness–tardiness scheduling on uniform parallel machines using simulated annealing and fuzzy logic approach. *Proceedings of the Institution of Mechanical Engineers, Part B : Journal of Engineering Manufacture*, v. 222, n. 2, p. 333 – 346, 2008.

[68] RIBEIRO, Amanda Gonçalves. "Função Linear"; Brasil Escola. Disponível em <https://brasilescola.uol.com.br/matematica/funcao-linear.htm>. Acesso em: 3 de junho de 2019.

[69] RINNOOY KAN, Alexander Hendrik George. Machine Scheduling Problems: Classification, complexity and computations. [S.l.]: Springer US, 1976. ISBN 9789024718481. DOI: 10.1007/978-1-4613-4383-7.

[70] RIOS-SOLIS, Y.; SOURD, F. Exponential neighborhood search for a parallel machine scheduling problem. *Computers & Operations Research*, v. 35, n. 5, p. 1697 – 1712. Part Special Issue: Algorithms and Computational Methods in Feasibility and Infeasibility, 2008.

[71] RODRIGUES, R.; PESSOA, A.; UCHOA, E.; DE ARAGÃO, M. P. Heuristic Algorithm for the Parallel Machine Total Weighted Tardiness Scheduling Problem. Relatório de Pesquisa em Engenharia de Produção, v. 8, n. 10, 2008.

[72] ROJAS, Alexander Javier Benavides. Heuristics for flow shop scheduling: considering non-permutation schedules and a heterogeneous workforce. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2015.

[73] ROLIM, José. Programação Linear. Pontifícia Universidade Católica Rio. Julho, 2010. Disponível em: <http://www-di.inf.puc-rio.br/ laber/LP2010-2.pdf>. Acesso em: 3 de junho de 2019.

[74] SCHALLER, J. E. Minimizing total tardiness for scheduling identical parallel machines with family setups. *Computers & Industrial Engineering*, v. 72, n. 0, p. 274 – 281, 2014.

[75] SHIM, S.-O.; KIM, Y.-D. Minimizing total tardiness in an unrelated parallel-machine scheduling problem. *Journal of the Operational Research Society*, v. 58, n. 3, p. 346 – 354, 2007a.

[76] SHIM, S.-O.; KIM, Y.-D. Scheduling on parallel identical machines to minimize total tardiness. *European Journal of Operational Research*, v. 177, n. 1, p. 135 – 146, 2007b.

[77] SILVA, M. M.; SUBRAMANIAN, A.; VIDAL, T.; OCHI, L. S. A simple and effective metaheuristic for the minimum latency problem. *European Journal of Operational Research*, v. 221, n. 3, p. 513 – 520, 2012.

[78] SIVRIKAYA-SERIFOGLU, F.; ULUSOY, G. Parallel machine scheduling with earliness and tardiness penalties. *Computers & Operations Research*, v. 26, n. 8, p. 773 – 787, 1999.

[79] STAFFORD, Edward F. On the Development of a Mixed-Integer Linear Programming Model for the Flowshop Sequencing Problem. The Journal of the Operational Research Society, v. 39, n. 12, p. 1163–1174, 1988. ISSN 0160-5682. DOI: 10.2307/2583601.

[80] SUBRAMANIAN, A.; BATTARRA, M. An iterated local search algorithm for the travelling salesman problem with pickups and deliveries. *Journal of the Operational Research Society*, v. 64, n. 3, p. 402–409, 2013.

[81] SUBRAMANIAN, A.; BATTARRA, M.; POTTS, C. N. An iterated local search heuristic for the single machine total weighted tardiness scheduling problem with

sequence-dependent setup times. *International Journal of Production Research*, v. 52, n. 9, p. 2729–2742, 2014.

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

[83] SUCUPIRA, Igor Ribeiro. MÉTODOS HEURÍSTICOS GENÉRICOS: METAHEURÍSTICAS E HIPER-HEURÍSTICAS. 2004. Dissertação – Departamento de Ciência da Computação, Universidade de São Paulo, São Paulo, 2004.

[84] SUN, H.; WANG, G. Parallel machine earliness and tardiness scheduling with proportional weights. *Computers & Operations Research*, v. 30, n. 5, p. 801 – 808, 2003.

[85] SZWARC, Wlodzimierz. The Flow-Shop Problem with Mean Completion Time Criterion. IIE Transactions, v. 15, n. 2, p. 172–176, jun. 1983. ISSN 0740-817X. DOI: 10.1080/05695558308974629.

[86] TAILLARD, E.; Benchmarks for basic scheduling problems. EJOR, v. 64, n. 2, p. 278-285, 1993.

[87] TANAKA, S.; ARAKI, M. A branch-and-bound algorithm with lagrangian relaxation to minimize total tardiness on identical parallel machines. *International Journal of Production Economics*, v. 113, n. 1, p. 446 – 458. Research and Applications in E-Commerce and Third-Party Logistics Management Special Section on Meta-standards in Operations Management: Cross-disciplinary perspectives, 2008a.

[88] TEIXEIRA, Márcio Andrey. Escalonamento de Processo. Universidade Federal de São Paulo. Disponível em: <http://ctd.ifsp.edu.br/ marcio.andrey/images/Escalonamento-Processos-IFSP.pdf>. Acesso em: 3 de junho de 2019.

[89] VALLADA, E.; RUIZ, R. Scheduling unrelated parallel machines with sequence dependent setup times and weighted earliness-tardiness minimization. RÍOS-MERCADO,R.Z.; RÍOS-SOLÍS, Y. A. (Eds.), $Just - in - Time\ Systems$, volume 60 of *Springer Optimization and Its Applications*, p. 67–90. Springer New York, 2012.

[90] VIDAL, T.; CRAINIC, T. G.; GENDREAU, M.; PRINS, C. Timing problems and algorithms: Time decisions for sequences of activities. *Networks*, v. 65, n. 2, p. 102–128, 2015.

[91] WEBSTER, S. New bounds for the identical parallel processor weighted flow time problem. Management Science, v. 38, n. 1, p. 124–136, 1992.

[92] WEBSTER, S. T. A priority rule for minimizing weighted flow time in a class of parallel machine scheduling problems. European Journal of Operational Research, v. 70, n. 3, p. 327–334, 1993.

[93] WEBSTER, S. Weighted flow time bounds for scheduling identical processors. European Journal of Operational Research, v. 80, n. 1, p. 103 – 111, 1995.

[94] WENG, M. X.; LU, J.; REN, H. Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, v. 70, n. 3, p. 215 – 226, 2001.

[95] XI, Y.; JANG, J.; FRIEDMAN, D.; HOU, W. A tardiness-concerned constructive method for the identical parallel machine scheduling. *The International Journal of Advanced Manufacturing Technology*, p. 1–12. ISSN 0268-3768, 2015.

[96] XU, X., XU, Z., GU, X.. An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization. *Expert Systems with Applications*, 38, 7970–7979, 2011.

[97] XU, Z,; XU, D.; HE, J.;WANG, Q.;LIU, A.;XIAO, J. Mixed Integer Programming Formulations for Two-Machine Flow Shop Scheduling with an Availability Constraint. *Arabian Journal for Science and Engineering*. v.43, p. 777-778, 2018.

[98] YALAOUI, F.; CHU, C. New exact method to solve the $P_m \mid r_j \mid c_j$ schedule problem. *International Journal of Production Economics*, v. 100, n. 1, p. 168–179, 2006.

[99] YALAOUI, F.; CHU, C. Parallel machine scheduling to minimize total tardiness. *International Journal of Production Economics*, v. 76, n. 3, p. 265 – 279, 2002.

[100] YING, Kuo-Ching; LIN, Shih-Wei. Minimizing makespan for the distributed hybrid flowshop scheduling problem with multiprocessor tasks. *Expert Systems with Applications*, vol. 92, p. 132-141, 2018.

[101] YOUSEFI, M.; YUSUFF, R. M. Minimising earliness and tardiness penalties in single machine scheduling against common due date using imperialist competitive algorithm. *International Journal of Production Research*, v. 51, n. 16, p. 4797–4804, 2013.

[102] ZHOU, H.; LI, Z.; WU, X. Scheduling unrelated parallel machine to minimize total weighted tardiness using ant colony optimization. *Automation and Logistics*, 2007 *IEEE International Conference on*, p. 132–136, 2007.

[103] ZHU, Z.; HEADY, R. B. Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. *Computers & Industrial Engineering*, v. 38, n. 2, p. 297 – 305, 2000.

[104] ŞEN, H.; Bülbül, K. A strong preemptive relaxation for weighted tardiness and earliness/tardiness problems on unrelated parallel machines. INFORMS Journal on Computing, v. 27, n. 1, p. 135–150, 2015.