UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinicius Lopes da Silva Teixeira

Algoritmo Iteração Gulosa com Busca Local em Soluções Parciais e Algoritmo Busca do Macaco Híbrida Aplicados ao Problema Flow Shop Permutacional

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinicius Lopes da Silva Teixeira

Algoritmo Iteração Gulosa com Busca Local em Soluções Parciais e Algoritmo Busca do Macaco Híbrida Aplicados ao Problema Flow Shop Permutacional

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito parcial exigido à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Alexsandro Santos Soares

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado Ciência da Computação

Uberlândia, Brasil

2018

Agradecimentos

Em primeiro lugar eu agradeço a Deus, agradeço pela graça derramada sobre a mim, pelas infindáveis bençãos, por me sustentar com Seu amor leal e por ter permitido com que eu concluísse mais essa etapa me dando sabedoria e o dom da vida a cada dia.

Aos meus pais Sebastião e Cleusa e aos meus irmãos Victor e Vanessa e aos meus familiares que não mediram esforços para que eu pudesse atingir o final da graduação, por todo o suporte, por acreditar em mim durante essa árdua caminhada.

A todos os professores que compartilharam seus conhecimentos e experiencias durante o meu processo de formação acadêmico, profissional e pessoal. Em especial eu agradeço ao meu orientador Alexandro por todo auxilio, incentivo, dedicação e paciência durante todo o período de pesquisa e escrita desta monografia.

Aos caros Thomas Stützle e M.K. Marichelvam por toda atenção, disponibilidade e prontidão em conversar comigo para responder as minhas dúvidas e questionamentos com relação aos algoritmos propostos em seus respectivos artigos originais.

A todos os meus amigos de longa data por todo apoio e incentivo. Aos amigos que fiz durante a graduação e a cada colega da turma 51 do curso de Ciência da Computação por fazerem parte desta caminhada, por todas as horas de estudo e risadas juntos.

A cada um dos meus amigos mais que especiais denominados Ferinhas por toda parceria e tempo juntos. Agradeço por todos os momentos de estudos e brincadeiras, de tristezas e alegrias, de dificuldades e vitórias, de desanimo e de animo em que atravessamos e nos fortaleceram.

Muito obrigado a todos que participaram e contribuíram de maneira direta ou indireta durante a conclusão desta importante etapa da minha vida.

“Porque Deus amou ao mundo tanto que deu seu único Filho Jesus Cristo, para que todo aquele que nEle crer não morra, mas tenha a vida eterna”
(Evangelho de João, capítulo 3 versículo 16; Bíblia Sagrada)

TEIXEIRA, Vinicius Lopes da Silva. **Algoritmo Iteração Gulosa com Busca Local em Soluções Parciais e Algoritmo Busca do Macaco Híbrida Aplicados ao Problema Flow Shop Permutacional**. 2018. 130p. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Federal de Uberlândia, Uberlândia, 2018.

Resumo

Este trabalho se refere ao Problema Flow Shop Permutacional (PFSP) com a função objetivo de minimizar o *makespan*. *Makespan* é uma medida de desempenho que calcula o tempo de conclusão do último trabalho a sair do sistema. O PFSP objetiva encontrar uma permutação de n trabalhos nos quais a ordem de processamento dos trabalhos dessa permutação é a mesma em todo o conjunto das m máquinas, de modo que todos os trabalhos passem por todas as máquinas. Neste trabalho foi realizada uma revisão sistemática da literatura na área do PFSP e a partir desta dois métodos foram escolhidos para serem analisados e comparados: a heurística Iteração Gulosa com Busca Local em Soluções Parciais (IG_{BLSP}) e a meta-heurística Busca do Macaco Híbrida (BMH). Para gerar a população inicial nos dois algoritmos, foi utilizada a heurística NEH, pelo fato de encontrar de maneira rápida resultados satisfatórios para o PFSP. O *benchmark* escolhido para os experimentos computacionais foi o desenvolvido por Taillard. Os resultados mostram que o IG_{BLSP} é eficiente, pois consegue encontrar boas soluções com um tempo de execução baixo. O BMH por sua vez encontrou soluções um pouco piores em comparação ao IG_{BLSP}, mas o principal ponto negativo deste método foi que ele apresentou um tempo de execução muito elevado ao encontrar as soluções, tornando-se assim ineficiente e inviável. Sendo assim, o IG_{BLSP} possui um custo-benefício amplamente superior ao BMH para resolver o PFSP minimizando o *makespan*.

Palavras-chave: Busca do Macaco; Flow Shop Permutacional; Iteração Gulosa; NEH; Revisão Sistemática da Literatura.

Abstract

This work refers to the Permutation Flow Shop Problem (PFSP) with the objective function of minimizing the makespan. Makespan is a performance measure that calculates the completion of the last that leaves the system. The PFSSP aims to find a permutation of n works which the processing order of tasks of this permutation is the same the whole set of machines, so that all tasks go through all machines. A systematic review of the PFSP literature was developed in this paper and the two following methods were chosen to be analyzed and compared: the heuristic Iterated Greedy with Optimization of Partial Solutions (IG_{OPS}) and the metaheuristic Hybrid Monkey Search (HMS). In order to generate the initial population in both algorithms, NEH heuristics were used, because it quickly found satisfactory results for the PFSSP. The benchmark chosen for the computational experiments was the develped by Taillard. Results show that IG_{OPS} is efficient because it can find good solutions with a low runtime. In other hand, HMS has found solutions slightly worse compared to IG_{OPS}, but the main negative point of this method was that it had a very high execution time when finding solutions, thus making it inefficient and unfeasible. So, the IG_{OPS} has a cost-benefit ratio that is broadly superior to the HMS to solve the PFSP minimizing the makespan.

Keywords: Monkey Search; Permutation Flow Shop; Iterated Greedy; NEH; Systematic Review of Literature.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

BLI	Busca Local Iterada
BMH	Busca do Macaco Híbrida
CDS	Campbell, Dudek e Smith
CAA	Colônia de Abelhas Artificiais
C_{max}	*Makespan*
DRP	Desvio Relativo Percentual
FSH	Flow Shop Híbrido
IG_{BLSP}	Iteração Gulosa com Busca Local em Soluções Parciais
LB	*Lower Bound*
MVP	Menor Valor de Posição
NEH	Nawaz, Enscore e Ham
OCF	Otimização em Colônia de Formigas
PFSP	Problema Flow Shop Permutacional
RSL	Revisão Sistemática da Literatura
UB	*Upper Bound*

Sumário

1 Introdução

Cada vez mais os problemas de otimização combinatória se evidenciam como uma importante área da computação. A ideia de um problema de otimização é bem simples: dada uma função objetivo, um conjunto de regras e algumas variáveis de decisão, escolha quais valores serão associados às variáveis de decisão utilizando o conjunto de regras de tal forma que a função objetivo atinja seu menor ou maior valor possível. A ideia pode ser bastante simples, mas a execução de tal tarefa pode se tornar impraticável, já que o número de soluções possíveis é muito grande. Os problemas de otimização possuem soluções ótimas (a melhor resposta existente possível) ou aproximadas (não é a melhor resposta existente).

Tecnicamente estes problemas são conhecidos como NP-difíceis, ou seja, são tão difíceis quanto os problemas mais difíceis em NP, sendo NP o acrônimo em inglês para Tempo Polinomial Não Determinístico [1], que segundo Leeuwen (1998) denota o conjunto de problemas que são decidíveis em tempo polinomial por uma máquina de Turing não-determinística. Existem diversos destes problemas com aplicação no mundo real, tais como mapeamento genético, escalonamento de tarefas, rotas de aviões, grade horária em escolas e universidades, projetos de circuitos integrados, posicionamento de satélites, roteamento de veículos, sistemas de distribuição de energia elétrica, etc.

Geralmente a resolução de um problema de otimização passa por duas etapas. A primeira consiste em modelar o problema e a segunda em resolver este modelo utilizando um algoritmo computacional. Diferentes algoritmos podem ser utilizados para resolver os problemas de otimização. Estes algoritmos podem ser classificados como exatos ou heurísticos.

As técnicas exatas encontram a melhor solução para um problema. Alguns exemplos de algoritmos exatos são: *branch & bound*, algoritmo simplex e programação dinâmica. Mas nem sempre a melhor solução será encontrada em um tempo razoável através dos algoritmos exatos, é justamente nesse momento que entram os métodos heurísticos. Estes métodos não garantem a solução ótima, mas eles entregam uma solução de qualidade em um bom tempo. Alguns exemplos de algoritmos heurísticos são: algoritmos genéticos, recozimento simulado [2], algoritmos meméticos [3], otimização em colônia de formigas e busca tabu [4].

[1] Tradução direta do inglês de *Non-Deterministic Polynomial time*
[2] Tradução direta do inglês de *Simulated Annealing*
[3] Tradução direta do inglês de *Memetic Algorithm*
[4] Tradução direta do inglês de *Tabu Search*

1.1 Questão de pesquisa

Partindo dos resultados descritos no artigo de Marichelvam, Tosun e Geetha (2017) em que se afirma ser a meta-heurística *Busca do Macaco Híbrida* melhor do que a maioria das implementações existentes na literatura e a implementação de uma melhoria do já difundido e eficiente algoritmo *Iteração Gulosa* presente em Dubois-Lacoste, Pagnozzi e Stützle (2017), será que ambos os algoritmos realmente apresentam bons resultados e são viáveis para a resolução do Problema *Flow Shop* Permutacional (PFSP)? Entre os dois qual apresenta um melhor custo-benefício?

1.2 Hipótese

Considerando a qualidade da solução, o IG_{BLSP} e o BMH encontram soluções próximas do *Upper Bound* para o PFSP utilizando o *benchmark* de Taillard. Com relação ao tempo de execução gasto na obtenção da solução, o IG_{BLSP} consome menos tempo quando comparado ao BMH para encontrar tais soluções.

1.3 Objetivos

Dado o problema *flow shop* permutacional, este trabalho tem como objetivo comparar a heurística IG_{BLSP} com a meta-heurística BMH de modo a verificar qual apresenta o melhor custo-benefício entre a qualidade da solução e o tempo de processamento.

1.3.1 Objetivos específicos

1. Comparar o resultado obtido a partir da implementação do IG_{BLSP} com o resultado original encontrado em Dubois-Lacoste, Pagnozzi e Stützle (2017);
2. Contrastar o resultado encontrado a partir da implementação do BMH com o resultado original encontrado em Marichelvam, Tosun e Geetha (2017);
3. Confrontar e analisar os resultados alcançados a partir das implementações dos algoritmos propostos entre si.

1.4 Justificativa

Em muitos setores, como por exemplo a industria farmacêutica, de eletrônicos, manufatura, metalúrgica, automotiva, cerâmica, linhas de produção, a forma com que se gerencia a produção irá definir se o ambiente é eficiente, se há desperdícios de recursos, se está em sua capacidade máxima, etc. Dessa forma, a gestão da produção sempre apresenta

problemas de otimização de trabalho, uma vez que a busca pela melhoria da eficiência dos processos passa necessariamente pelo incremento de alguma medida de desempenho.

Por este motivo, o estudo e a aplicação dos algoritmos de otimização combinatória são de grande importância, pois possibilitam que um problema de otimização seja conhecido a fundo em suas dificuldades e possa ser resolvido da melhor maneira possível. Os algoritmos que serão vistos aqui terão a finalidade de mostrar qual é a melhor solução para o problema *flow shop* permutacional atualmente.

1.5 Repetibilidade em otimização combinatória

De acordo com Collberg e Proebsting (2016), existem duas razões principais para compartilhar os artefatos de uma pesquisa: repetibilidade e beneficiamento. Uma pesquisa é repetível quando se pode executar novamente o experimento dos pesquisadores usando a mesma metodologia, no mesmo ambiente e obtendo os mesmos resultados. A repetibilidade garante que colegas e revisores possam avaliar os resultados com base em evidências completas e precisas. O beneficiamento permite que os colegas e pesquisadores usufruam dos resultados, melhorando o progresso científico evitando a replicação desnecessária de trabalho.

Ao contrário da repetibilidade, a reprodutibilidade não requer necessariamente o acesso aos artefatos originais da pesquisa. Pelo contrário, ela é a confirmação de uma hipótese científica feita de maneira independente, realizada após a publicação, coletando diferentes dados de diferentes experimentos executados em diferentes *benchmarks* e usando esses dados para verificar as afirmações feitas no artigo.

A repetibilidade e reprodutibilidade são pedras angulares, ou seja, fundamentais no processo científico, necessárias para evitar que resultados errôneos e falsos sejam disseminados.

A repetibilidade é de suma importância para as pesquisas em otimização combinatória, uma vez que a qualidade da solução encontrada e o tempo de execução dos algoritmos, que são os principais pilares desse tipo de pesquisa, devem ser fiéis àqueles encontrados nos trabalhos originais. A discrepância entre os resultados obtidos por uma pesquisa original e os resultados obtidos por meio da reprodução desta pesquisa pode levar a inutilização ou até mesmo a invalidação de tal pesquisa.

Não só em otimização combinatória, mas no geral como disciplina, a área da computação está muito longe de produzir pesquisas que sejam sempre e completamente repetíveis; para começar, no entanto, a mudar este cenário, poderia se exigir que os autores informem de maneira consciente à comunidade sobre suas intenções em relação ao compartilhamento de seus artefatos de pesquisa. Esta informação deve ser fornecida pelos autores ao submeter

o seu trabalho para publicação. Isso permitiria aos revisores considerar se a pesquisa atingiu um nível esperado de repetibilidade e assim determinar se aceitam ou rejeitam o trabalho.

Um dos problemas frequentes em pesquisas na área da computação é que as publicações por vezes não disponibilizam o código fonte que respalda a mesma, o que dificulta ou mesmo impede a repetição dos experimentos pelos pares científicos. Para melhorar o estado de repetibilidade nas pesquisas em computação, poderia simplesmente ser exigido aos autores anexarem o código fonte correspondente a pesquisa, juntamente com todos os trabalhos submetidos para publicação.

Infelizmente, com base na pesquisa realizada por Collberg e Proebsting (2016), é irreal esperar que os pesquisadores da área de computação sempre disponibilizem seu código fonte para outras pessoas. Existem várias razões para isso: o código pode não estar limpo o suficiente para distribuição pública; os autores podem estar considerando a comercialização; o código (ou parte dele) pode ter restrições de licenciamento; os autores podem estar ocupados demais para responder a perguntas sobre seu sistema ou eles podem não querer fazer qualquer trabalho de acompanhamento.

1.6 Organização do Trabalho

O capítulo 2 introduz os principais conceitos sobre o escalonamento de tarefas e também sobre o PFSP.

O capítulo 3 contém a Revisão Sistemática da Literatura. Além da fundamentação da RSL, estão descritos o protocolo de pesquisa e os resultados obtidos através da pesquisa realizada.

O capítulo 4 apresenta todos os detalhes e passos para a implementação dos algoritmos NEH, IG_{BLSP} e BMH. Também foi demonstrado um exemplo numérico para cada um dos três métodos.

O capítulo 5 detalha o desenvolvimento obtido neste trabalho. Nele foi apresentado o ambiente de execução, as instâncias utilizadas para análise e foi realizada uma discussão sobre os resultados obtidos através dos experimentos computacionais.

O capítulo 6 discorre sobre as conclusões obtidas por meio dos resultados e análises encontrados durante a monografia, mostra as contribuições aqui alcançadas e exibe os trabalhos futuros a serem realizados.

2 Fundamentação Teórica

O problema tratado neste trabalho está inserido na classe de escalonamento de tarefas (ou escalonamento de processos), a qual consiste em determinar a ordem ou a sequência em que máquinas irão processar trabalhos de modo a otimizar alguma função objetivo (medida de desempenho).

Este problema foi proposto primeiramente por Johnson (1954) e desde então, de acordo com Hordones, Camargo e Fuchigami (2016), tornou-se uma área de pesquisa que vem sendo bastante estudada por pessoas de pesquisa operacional, de produção e operações e por matemáticos. Nas últimas 6 décadas uma grande quantidade de trabalhos que abordam os problemas de escalonamento foram desenvolvidos empregando inúmeras configurações de ambientes de máquinas, distintas restrições e diversas funções objetivo.

Um exemplo clássico de escalonamento de tarefas é uma linha de fabricação de placas de circuito, um problema do mundo real naturalmente expresso como um PFSP. As esteiras onde as placas são montadas correspondem às máquinas e as placas de circuito correspondem aos trabalhos executados em cada máquina. Placas de circuito maiores tendem a ter mais componentes e podem exigir tempos de processamento maiores em cada esteira na linha de montagem. A execução dessas tarefas estão correlacionadas aos tempos de processamento nas máquinas. Algumas medidas de desempenho para este caso podem ser a redução do tempo de processamento, diminuir os gastos em energia elétrica e a minimização do atraso máximo de entrega das placas.

2.1 Escalonamento de tarefas

Existem várias estruturas, notações e hierarquias de complexidade para os problemas de escalonamento. Nesta seção foram apresentadas e detalhadas as classificações mais comuns presentes na literatura para os diversos problemas de escalonamento de tarefa existentes.

2.1.1 Estrutura e notação

Em todos os problemas de escalonamento de tarefas, assume-se que o número de máquinas e de trabalhos são finitos. O número de trabalhos é denotado por n, enquanto o número de máquinas é denotado por m. A representação de um trabalho é dada pelo índice j e de uma máquina pelo índice i. O par (i,j) se refere a um passo de processamento ou operação do trabalho j sobre a máquina i. De acordo com Pinedo (2012), os seguintes dados são associados ao trabalho j:

Tempo de processamento [1] (p_{ij}): representa o tempo de processamento de um trabalho j em uma máquina i. O índice i pode ser omitido se o tempo de processamento do trabalho j não depende da máquina i ou se o trabalho j é processado somente em uma determinada máquina.

Data de liberação [2] (r_j): é o momento em que o trabalho j chega ao sistema, ou seja, é o primeiro instante em que o trabalho pode começar seu processamento.

Data de Vencimento [3] (d_j): representa a data de conclusão do trabalho j, isto é, a data em que o trabalho é prometido ao cliente. É permitida a conclusão de um trabalho após a data de vencimento, no entanto uma penalidade será aplicada. Quando uma data de vencimento *deve* ser cumprida, então ela recebe o nome de *prazo* e é denotada por $\overline{d}_j$.

Peso [4] (w_j): basicamente é um fator de prioridade, que denota a importância (peso) de um trabalho j em relação aos outros trabalhos do sistema. Quanto maior o peso, maior é a prioridade dada ao trabalho.

Os problemas de escalonamento de tarefas podem ser classificados de acordo com a seguinte tripla $\alpha \mid \beta \mid \gamma$. O campo α descreve o ambiente em que a máquina esta inserida e contém apenas uma entrada. O campo β fornece detalhes de características e restrições de processamento e pode conter uma, múltiplas ou nenhuma entrada. O campo γ descreve a função objetivo (medida de desempenho) a ser minimizada e geralmente contém uma única entrada.

2.1.1.1 Campo α: ambiente de máquina

Os possíveis ambientes em que uma máquina pode ser inserida são:

Máquina única [5] (1): é o caso mais simples de todos os ambientes de máquinas e é também um caso especial de todos os outros ambientes de máquinas mais complicados.

Máquinas idênticas em paralelo [6] (Pm): existem m máquinas idênticas em paralelo. Um trabalho j simboliza uma única operação e pode ser processado em qualquer uma das m máquinas ou em qualquer uma que pertença a um determinado subconjunto. A entrada M_j aparece no campo β se o trabalho j não puder ser processado em uma máquina qualquer, mas apenas em alguma pertencente a um subconjunto específico M_j.

Máquinas em paralelo com velocidades diferentes [7] (Qm): existem m máquinas em paralelo com velocidades diferentes. A velocidade da máquina i é representada por v_i. O tempo p_{ij} que o trabalho j demora na máquina i é igual a (p_j / v_i) (assumindo que o trabalho

[1] Tradução direta do inglês de *Processing time*
[2] Tradução direta do inglês de *Release date*
[3] Tradução direta do inglês de *Due date*
[4] Tradução direta do inglês de *Weight*
[5] Tradução direta do inglês de *Single machine*
[6] Tradução direta do inglês de *Identical machines in parallel*
[7] Tradução direta do inglês de *Machines in parallel with different speeds*

j recebe todo o seu processamento da máquina i). Este ambiente é referido como máquinas *uniformes*. Se todas as máquinas tiverem a mesma velocidade, isto é, $v_i = 1$ para todos i e $p_{ij} = p_j$, então o ambiente é idêntico ao anterior.

Máquinas não relacionadas em paralelo [8] (Rm) : este ambiente é uma generalização adicional do anterior. Existem várias máquinas diferentes em paralelo. A máquina i pode processar o trabalho j na velocidade v_{ij}. O tempo p_{ij} que o trabalho j gasta na máquina i é igual a (p_j / v_{ij}) (novamente assumindo que o trabalho j recebe todo o seu processamento da máquina i). Se as velocidades das máquinas i forem independentes dos trabalhos, ou seja, $v_{ij} = v_i$ para todos i e j, então o ambiente é idêntico ao anterior.

Flow shop (Fm): existem m máquinas em série. Cada trabalho j deve passar em cada uma das m máquinas deste ambiente. Todos os trabalhos seguem a mesma rota. Depois de completar seu processamento em uma máquina, o trabalho vai para a fila da próxima máquina. Usualmente todas as filas assumem operações *Primeiro a Entrar, Primeiro a Sair* [9] (PEPS), isto é, um trabalho nunca pode "passar" na frente de outro enquanto espera na fila. Se o PEPS for eficiente no *flow shop*, então o problema será chamado *flow shop* permutacional e o campo β vai incluir a entrada *prmu*.

Flow shop híbrido (FFc): é uma generalização do *flow shop* e do ambiente de máquinas paralelas. Em vez de m máquinas em série, existem c estágios em série. Cada estágio possui um número de máquinas idênticas em paralelo, este número de máquinas pode ser o mesmo em cada estágio ou pode ser um número diferente em cada estágio. Cada trabalho deve ser processado primeiramente no estágio 1, depois no estágio 2, e assim por diante. Um estágio funciona como um banco de máquinas paralelas e em cada um deles o trabalho j é processado em apenas uma máquina e qualquer máquina pode fazer tal processamento. As filas entre os vários estágios podem ou não funcionar de acordo com a disciplina *Primeiro a Chegar, Primeiro a ser Servido* [10] (PCPS). (O *flow shop* híbrido também pode ser encontrado na literatura com o nome *flow shop* flexível).

Job shop (Jm): no *job shop* com m máquinas, cada trabalho tem sua própria rota predeterminada que deve seguir. É feita uma distinção entre os *job shops* em que cada trabalho visita cada máquina no máximo uma vez e os *job shops* em que um trabalho pode visitar cada máquina mais de uma vez. Neste último caso, o campo β contém a entrada *rcrc* que significa *recirculação*.

Job shop híbrido (FJc): é uma generalização do *job shop* e do ambiente de máquinas paralelas. Em vez de m máquinas em série, existem c Centros de Trabalho (CT). Cada CT possui um número de máquinas idênticas em paralelo, este número de máquinas pode ser o mesmo em cada CT ou pode ser um número diferente. Cada trabalho tem sua própria

[8] Tradução direta do inglês de *Unrelated machines in parallel*
[9] Tradução direta do inglês de *First In First Out*
[10] Tradução direta do inglês de *First Come First Served*

rota a seguir; em cada CT, o trabalho *j* será processado em apenas uma máquina e qualquer máquina pode fazer tal processamento. Se um trabalho em sua rota pode visitar um centro de trabalho mais de uma vez, então o campo β contém a entrada *rcrc* que significa *recirculação*. (O *job shop* híbrido também pode ser encontrado na literatura com o nome *job shop* flexível).

Open shop (Om): existem m máquinas. Cada trabalho deve ser processado em cada uma das máquinas m. No entanto, alguns desses tempos de processamento podem ser zero. Não há restrições quanto a rota de cada trabalho através do ambiente de máquina. O programador pode determinar uma rota para cada trabalho e diferentes trabalhos podem ter rotas diferentes.

2.1.1.2 Campo β: características e restrições de processamento

As características específicas e restrições de processamento podem conter uma, nenhuma ou múltiplas entradas. Algumas possíveis entradas em β são:

Data de liberação [11] (r_j): se esse símbolo aparecer no campo β, o trabalho j não pode iniciar seu processamento antes da data de liberação r_j. Se r_j não aparecer no campo β, o processamento do trabalho j pode começar a qualquer momento. Diferentemente das datas de liberação, as datas de vencimento não são especificadas neste campo, pois o tipo de função objetivo dá indicações suficientes, quer existam datas de vencimento ou não.

Preempção [12] $(prmp)$: implica que não é necessário manter um trabalho em uma máquina, uma vez iniciado, até sua conclusão. O programador pode interromper o processamento de um trabalho (antecipar) em qualquer momento e colocar um trabalho diferente na máquina. A quantidade de processamento que um trabalho antecipado já recebeu não é perdida. Quando um trabalho antecipado é posto de volta na máquina (ou em outra máquina no caso de máquinas paralelas), ele só precisa da máquina para o tempo de processamento restante. Quando são permitidas preempções, o *prmp* é incluído no campo β; Quando o $prmp$ não está incluído, então a preempção não é permitida.

Restrições de precedência [13] $(prec)$: podem aparecer em uma única máquina ou em um ambiente de máquinas paralelas, exigindo que um ou mais trabalhos tenham que ser concluídos antes que outro trabalho possa iniciar seu processamento. Existem várias formas especiais de restrições de precedência: se cada trabalho tem no máximo um antecessor e, no máximo, um sucessor, as restrições são referidas como *cadeias*. Se cada trabalho tem no máximo um sucessor, as restrições são referidas como um *intree*. Se cada trabalho tem no máximo um antecessor, as restrições são referidas como um *outtree*. Se nenhum $prec$ aparecer no campo β, os trabalhos não estão sujeitos a restrições de precedência.

[11] Tradução direta do inglês de *Release date*
[12] Tradução direta do inglês de *Preemptions*
[13] Tradução direta do inglês de *Precedence constraints*

Tempo de preparação dependente da sequência [14] (s_{jk}): representa o tempo de preparação dependente da sequência que é incorrido entre o processamento dos trabalhos j e k; s_{0k} indica o tempo de preparação para o trabalho k se este for o primeiro na sequência e s_{j0} indica o tempo de limpeza após o trabalho j se o trabalho j for último na sequência (é claro, s_{0k} e s_{j0} podem ser zero). Se o tempo de configuração entre os trabalhos j e k depende da máquina, então o subíndice i será incluído, isto é, s_{ijk}. Se nenhum s_{jk} aparecer no campo β, todos os tempos de preparação são assumidos como 0 ou independentes da sequência (caso em que são simplesmente incluídos nos tempos de processamento).

Famílias de trabalho [15] $(fmls)$: os n trabalhos pertencem, neste caso, a F famílias diferentes de trabalho. Os trabalhos da mesma família podem ter tempos de processamento diferentes, mas podem ser processados em uma mesma máquina um após o outro sem requerer qualquer preparação entre eles. No entanto, se a máquina mudar de uma família para outra, por exemplo, da família g para a família h, então uma preparação é necessária. Se este tempo de preparação depende de ambas as famílias g e h e depende da sequência, então é indicado por s_{gh}. Se este tempo de preparação depende apenas da família que está prestes a começar, ou seja, a família h, então é denotado por s_h. Se não depende de nenhuma das duas famílias, é indicado por s.

Processamento em lote [16] $(batch(b))$: uma máquina pode processar vários trabalhos, digamos b, simultaneamente; ou seja, pode processar um lote de até b trabalhos ao mesmo tempo. Os tempos de processamento dos trabalhos em um lote podem não ser todos iguais e todo o lote é concluído somente quando o último trabalho do lote for concluído, o que implica que o tempo de conclusão do lote inteiro é determinado pelo trabalho com o maior tempo de processamento. Se $b = 1$, o problema se reduz a um ambiente de escalonamento convencional. Outro caso especial que é interessante é $b = \infty$, isto é, não há limite no número de trabalhos que a máquina pode manipular a qualquer momento.

Quebra [17] $(brkdwn)$: as quebras de máquinas implicam que uma máquina pode não estar disponível continuamente. Os períodos em que uma máquina não está disponível são, assumidos como corrigíveis (por exemplo, devido a mudanças ou manutenção programada). Se houver uma série de máquinas idênticas em paralelo, o número de máquinas disponíveis em qualquer ponto no tempo é uma função do tempo, isto é, $m(t)$. As quebras de máquina às vezes também são chamadas de restrições de disponibilidade da máquina.

Restrições de elegibilidade da máquina [18] (M_j): o símbolo M_j pode aparecer no campo β quando o ambiente de máquina contém m máquinas em paralelo (Pm). Quando o M_j está presente, nem todas as m máquinas são capazes de processar o trabalho j. Definir

[14] Tradução direta do inglês de *Sequence dependent setup times*
[15] Tradução direta do inglês de *Job families*
[16] Tradução direta do inglês de *Batch processing*
[17] Tradução direta do inglês de *Breakdowns*
[18] Tradução direta do inglês de *Machine eligibility restrictions*

M_j é o mesmo que indicar o conjunto de máquinas que podem processar o trabalho *j*. Se o campo β não contém M_j, o trabalho *j* pode ser processado em qualquer uma das máquinas *m*.

Permutação [19] $(prmu)$: uma restrição que pode aparecer no ambiente *flow shop* é que as filas de processamento de cada máquina operam de acordo com a disciplina *First In First Out* (*FIFO*). Isso implica que a ordem (ou permutação) na qual os trabalhos passam pela primeira máquina é mantida em todo o restante do sistema.

Bloqueio [20] $(block)$: é um fenômeno que pode ocorrer no *flow shop*. Se um *flow shop* tiver um *buffer* limitado entre duas máquinas sucessivas, pode acontecer que, quando o *buffer* estiver cheio, a máquina atual não poderá liberar um trabalho concluído. O bloqueio implica que o trabalho completo deve permanecer na máquina atual impedindo (ou seja, bloqueando) que a máquina trabalhe no próximo trabalho. A ocorrência mais comum de bloqueio é o caso com zero *buffers* entre duas máquinas sucessivas. Nesse caso, um trabalho que concluiu seu processamento em uma determinada máquina não pode deixar a máquina se o trabalho anterior ainda não tiver concluído seu processamento na próxima máquina; assim, o trabalho bloqueado também impede (ou bloqueia) o próximo trabalho de iniciar seu processamento na máquina fornecida.

Sem espera [21] (nwt): este é outro fenômeno que pode ocorrer no *flow shop*. Os trabalhos não podem esperar entre duas máquinas sucessivas. Isso implica que o tempo de início de um trabalho na primeira máquina deve ser adiado para garantir que o trabalho possa passar pelo *flow shop* sem ter que esperar por qualquer máquina.

Recirculação [22] $(rcrc)$: ocorre em um *job shop* ou em um *job shop híbrido* quando um trabalho pode visitar uma máquina ou um centro de trabalho mais de uma vez.

2.1.1.3 Campo γ: função objetivo

O campo γ descreve a função objetivo (medida de desempenho) a ser minimizada e geralmente contém uma única entrada. Segundo Pinedo (2012) o objetivo a ser minimizado é sempre uma função dos tempos de conclusão dos trabalhos, que dependem do escalonamento. O tempo de conclusão da operação do trabalho *j* na máquina *i* é denotado por C_{ij}. O tempo em que o trabalho *j* sai do sistema (ou seja, o tempo de conclusão na última máquina na qual ele requer processamento) é denotado por C_j.

O *atraso* do trabalho *j* é definido pela equação

$$L_j = C_j - d_j,$$

[19] Tradução direta do inglês de *Permutation*
[20] Tradução direta do inglês de *Blocking*
[21] Tradução direta do inglês de *No-wait*
[22] Tradução direta do inglês de *Recirculation*

ou seja, será positivo se o trabalho *j* for concluído com atraso e será negativo se for concluído com antecedência.

A unidade de *penalidade* do trabalho *j* é definida como:

$$U_j = \begin{cases} 1 \text{ se } C_j > d_j \\ 0 \text{ se } C_j \leq d_j \end{cases}$$

As medidas de desempenho mais usuais são:

Makespan (C_{max}): definido como sendo $\max(C_1, ..., C_n)$, é equivalente ao tempo de conclusão do último trabalho a sair do sistema. Um *makespan* mínimo geralmente implica em uma boa utilização da(s) máquina(s).

Atraso máximo [23] (L_{max}): é definido como $\max(L_1, ..., L_n)$. Ele mede a pior violação das datas de vencimento.

Tempo total ponderado de conclusão [24] $(\sum w_j C_j)$: a soma ponderada dos tempos de conclusão dos *n* postos de trabalho dá uma indicação dos custos totais de exploração ou de inventário incorridos pelo escalonamento. A soma dos tempos de conclusão está muitas vezes referida na literatura como o tempo de fluxo. O tempo total ponderado de conclusão é então referido como o tempo ponderado de fluxo.

Tempo total ponderado de conclusão descontado [25] $(\sum w_j(1 - e^{-rC_j}))$: esta é uma função de custo mais geral do que a anterior, onde os custos são descontados a uma taxa de r, $0 < r < 1$, por unidade de tempo. Ou seja, se o trabalho *j* não for concluído no tempo *t*, um custo adicional $w_j r e^{-rt} dt$ será incorrido sobre o período $[t, t + dt]$. Se o trabalho *j* estiver concluído no tempo *t*, o custo total incorrido durante o período $[0, t]$ será $w_j(1 - e^{-rt})$. O valor de *r* é geralmente próximo de 0, digamos 0.1 ou 10%.

Número ponderado de trabalhos atrasados [26] $(\sum w_j U_j)$: não é apenas uma medida de interesse acadêmico, muitas vezes é um objetivo na prática,ou seja, no mundo real, pois é uma medida que pode ser registrada com muita facilidade.

2.1.2 Hierarquia de Complexidade

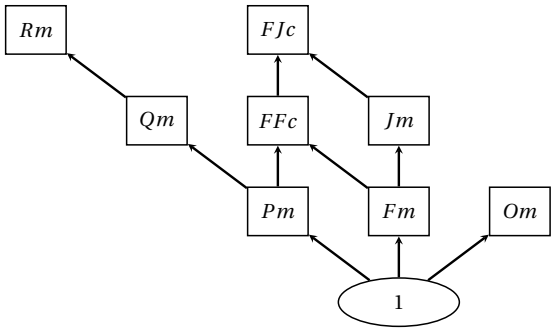
De acordo com Pinedo (2012), um esforço considerável foi feito para estabelecer uma hierarquia descrevendo as relações entre as centenas de problemas de escalonamento. Nas comparações entre as complexidades dos diferentes problemas de escalonamento, é importante saber que uma alteração em um único elemento na classificação de um problema afeta sua complexidade.

[23] Tradução direta do inglês de *Maximum Lateness*
[24] Tradução direta do inglês de *Total weighted completion time*
[25] Tradução direta do inglês de *Discounted total weighted completion time*
[26] Tradução direta do inglês de *Weighted number of tardy jobs*

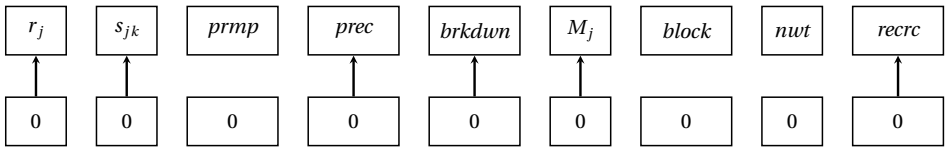
Na Figura 1, são exibidos vários gráficos que ajudam a determinar a hierarquia de complexidade dos problemas de escalonamento, sendo que a direção da seta indica que está saindo de um α, β ou γ de complexidade menor (base da seta) para um de complexidade maior (ponta da seta). A maioria da hierarquia descrita nas figuras é relativamente simples.

Figura 1 – Hierarquia de complexidade dos problemas de escalonamento.

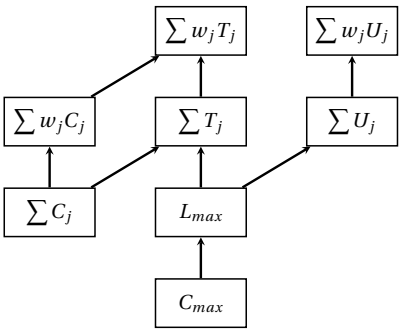
(a) Ambiente de máquina



(b) Características e restrições de processamento



(c) Função objetivo



FONTE: Adaptado de Pinedo (2012)

Pinedo (2012) afirma ainda que uma quantidade significativa de pesquisa em escalonamento foi dedicada a encontrar eficientemente (denominado tempo polinomial) algoritmos para problemas de escalonamento. No entanto, muitos destes problemas não possuem um algoritmo de tempo polinomial e são chamados de problemas *NP-difíceis* [27].

Verificar que um problema é NP-difícil requer uma prova matemática formal, e esta pode ser encontrada em Pinedo (2012). Antigamente as pesquisas na área de escalonamento

[27] Tradução direta do inglês de *NP-hard*

centravam-se, em particular, na fronteira entre os problemas solucionáveis em tempo polinomial e os problemas NP-difíceis. Os exemplos a seguir ilustram as fronteiras entre os problemas fáceis e difíceis em determinados conjuntos de problemas.

2.1.2.1 Exemplo de uma hierarquia de complexidade: C_{max}

Considerando os seguintes problemas:

a) $1 \,||\, C_{max}$ *(máquina única minimizando o makespan)*

b) $P2 \,||\, C_{max}$ *(duas máquinas idênticas em paralelo minimizando o makespan)*

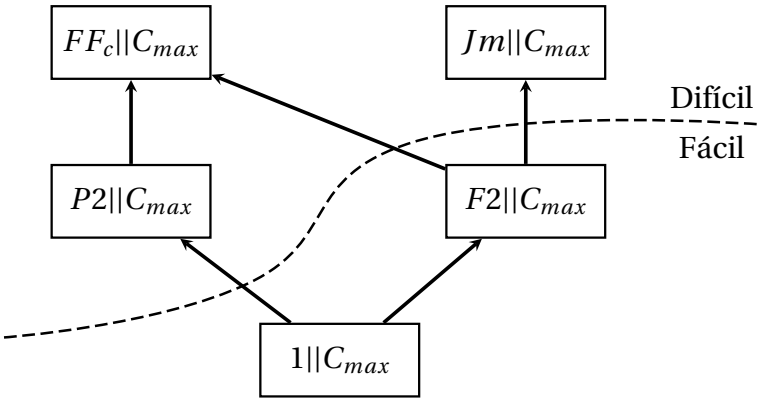
c) $F2 \,||\, C_{max}$ *(flow shop com duas máquinas em série minimizando o makespan)*

d) $Jm \,||\, C_{max}$ *(job shop com m máquinas minimizando o makespan)*

e) $FFc \,||\, C_{max}$ *(flow shop híbrido com c estágios minimizando o makespan)*

Na Figura 2 retrata-se a hierarquia desses problemas com relação a complexidade.

Figura 2 – Hierarquia de complexidade: C_{max}.



FONTE: Adaptado de Pinedo (2012)

2.1.2.2 Exemplo de uma hierarquia de complexidade: L_{max}

Dados os problemas a seguir:

a) $1 \,||\, L_{max}$ *(máquina única, minimizando o atraso máximo)*

b) $1 \,|\, prmp \,|\, L_{max}$ *(máquina única, preemptiva, minimizando o atraso máximo)*

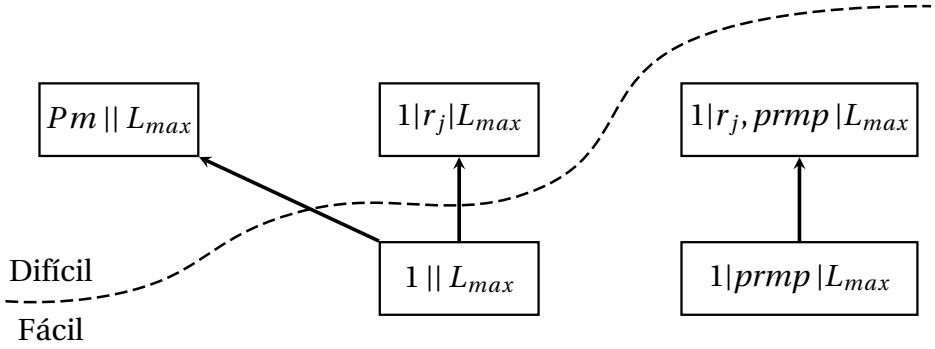
c) $1 \,|\, r_j \,|\, L_{max}$ *(máquina única, data de liberação, minimizando o atraso máximo)*

d) $1 \,|\, r_j, prmp \,|\, L_{max}$ *(máquina única, com data de liberação e preemptiva, minimizando o atraso máximo)*

e) $Pm \,||\, L_{max}$ *(m máquinas idênticas em paralelo, minimizando o atraso máximo)*

Na Figura 3 descreve-se a hierarquia desses problemas com relação a complexidade.

Figura 3 – Hierarquia de complexidade: L_{max}.



FONTE: Adaptado de Pinedo (2012)

2.2 O problema *flow shop* permutacional

De forma mais específica, nesta monografia será abordado o **Problema *Flow Shop* Permutacional** (PFSP) com o objetivo de **minimizar o *makespan***. Então pela notação mostrada na seção 2.1.1 este problema pode ser descrito como: $Fm \mid prmu \mid C_{max}$.

O PFSP tem como objetivo encontrar uma permutação $\pi = [\pi_1, \pi_2, ..., \pi_n]$ de n trabalhos nos quais a ordem de processamento dos trabalhos desta permutação é a mesma em todo o conjunto das m máquinas (isto é, não é permitido que os trabalhos troquem de ordem entre as máquinas), de modo que todos os trabalhos passem por todas as máquinas.

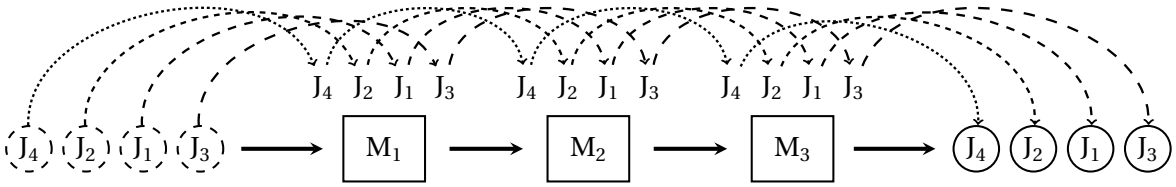
Como este é um problema permutacional, existem ***n!*** possíveis sequências de trabalhos para encontrar a melhor, com a finalidade de minimizar ou maximizar alguma medida de desempenho. No caso deste trabalho, o objetivo é minimizar o *makespan* (C_{max}).

De acordo com Yenisey e Yagmahan (2014) o PFSP tem os seguintes pressupostos:

i. Cada trabalho j só pode ser processado em uma máquina i ao mesmo tempo;
ii. Cada máquina i pode processar apenas um trabalho j ao mesmo tempo;
iii. É não preemptivo, ou seja, o processamento não pode ser interrompido;
iv. Todos os trabalhos são independentes e estão disponíveis para processamento no instante "zero";
v. Os tempos de preparação dos trabalhos nas máquinas estão incluídos no tempo de processamento p_{ij};
vi. As máquinas estão continuamente disponíveis.

Um ambiente de produção *flow shop* permutacional com 3 máquinas e 4 trabalhos pode ser visto na Figura 4, onde $\pi = [J_4, J_2, J_1, J_3]$, ou seja, os trabalhos serão sempre executados na seguinte ordem em cada máquina: 1°: $J_4 \longrightarrow$ 2°:$J_2 \longrightarrow$ 3°: $J_1 \longrightarrow$ 4°: J_3. Assim como os trabalhos, as máquinas que executam os trabalhos também tem uma ordem de processamento: 1°: $M_1 \longrightarrow$ 2°:$M_2 \longrightarrow$ 3°: M_3.

Figura 4 – Ilustração de um *flow shop* permutacional.



FONTE: Adaptado de Hordones, Camargo e Fuchigami (2016)

2.2.1 Definição Matemática

Dada uma permutação $j_1, j_2, ..., j_n$ para as m máquinas e os n trabalhos no ambiente *flow shop* permutacional, o tempo de conclusão C_{i,j_x} de um trabalho qualquer j_x em uma máquina i pode ser calculado usando o seguinte conjunto de equações:

$$C_{i,j_1} = \sum_{h=1}^{i} p_{h,j_1}, \quad \text{para } i = 1,2,...,m \tag{2.1}$$

$$C_{1,j_x} = \sum_{h=2}^{x} p_{1,j_h}, \quad \text{para } x = 2,...,n \tag{2.2}$$

$$C_{i,j_x} = max\left(C_{i-1,j_x}, C_{i,j_{x-1}}\right) + p_{i,j_x}, \quad \text{para } i = 2,...,m \quad \text{e} \quad x = 2,...,n \tag{2.3}$$

A equação 2.1 está somando apenas o tempo de processamento do primeiro trabalho em cada máquina, ou seja, esta construindo a primeira coluna dos tempos de conclusão.

A equação 2.2 está somando o tempo de processamento dos trabalhos apenas na primeira máquina, ou seja, esta construindo a primeira linha dos tempos de conclusão.

A equação 2.3 está calculando o tempo de processamento dos trabalhos nas máquinas restantes, em que o tempo de conclusão do atual trabalho na atual máquina é a soma do tempo de processamento do trabalho atual com o maior dentre o tempo de conclusão do mesmo trabalho na máquina anterior e o tempo de conclusão do trabalho anterior na mesma máquina.

Como visto na seção 2.1.1.3, o *makespan* $\left(C_{max}\right)$ é definido pelo último trabalho a deixar o ambiente, ou seja, $C_{max} = C_{m,j_n}$.

2.2.2 Exemplo PFSP

Consideremos um pequeno exemplo com 4 máquinas e 3 trabalhos e os seguintes tempos de processamento:

Tabela 1 – Exemplo numérico: Tempos de processamento para o PFSP.

		Trabalhos		
		J_1	J_2	J_3
Máquinas	M_1	54	83	15
	M_2	79	3	11
	M_3	16	89	49
	M_4	66	58	31

No total teremos 6 possíveis soluções, pois o número de permutações que podem ser encontradas é igual a $n!$, ou seja, $3! = 6$.

Uma das possíveis permutações é $\pi = [J_1, J_2, J_3]$. A Tabela 2 mostra o cálculo dos tempos de conclusão para esta permutação seguindo os tempos de processamento encontrados na Tabela 1. O cálculo dos tempos de processamento de todas as 6 permutações pode ser encontrado na Tabela 3.

Tabela 2 – Exemplo numérico: Tempos de conclusão para a permutação $\pi = [J_1, J_2, J_3]$.

(a) Equação 2.1: C_{i,j_1}

		Trabalhos		
		J_1	J_2	J_3
Máquinas	M_1	54	-	-
	M_2	133	-	-
	M_3	149	-	-
	M_4	215	-	-

(b) Equação 2.2: C_{1,j_x}

		Trabalhos		
		J_1	J_3	J_2
	M_1	-	137	152
	M_2	-	-	-
	M_3	-	-	-
	M_4	-	-	-

(c) Equação 2.3: C_{i,j_x}

		Trabalhos		
		J_2	J_1	J_3
	M_1	-	-	-
	M_2	-	140	163
	M_3	-	238	287
	M_4	-	296	327

2.2.3 Estado da Arte

O PFSP é um dos problemas mais estudados na literatura de otimização combinatória, sendo que a grande maioria dos pesquisadores da área escolhem como medida de desempenho a minimização do *makespan* e algoritmos aproximados, pois estes acham boas soluções com um tempo de resposta aceitável.

O trabalho de Ruiz e Maroto (2005) analisou 25 métodos heurísticos e meta-heurísticos testados sobre o *benchmark* de Taillard (1993) até 2004. A heurística NEH descrita em Nawaz,

Tabela 3 – Exemplo numérico: Cálculo do C_{max} a partir de todas as possíveis permutações dos trabalhos J_1, J_2, J_3.

(a) $J_1 - J_2 - J_3$

		Trabalhos		
		J_1	J_2	J_3
Máquinas	M_1	54	137	152
	M_2	133	140	163
	M_3	149	238	287
	M_4	215	296	327
		C_{max} = 327		

(b) $J_1 - J_3 - J_2$

		Trabalhos		
		J_1	J_3	J_2
	M_1	54	69	152
	M_2	133	144	155
	M_3	149	198	287
	M_4	215	246	345
		C_{max} = 345		

(c) $J_2 - J_1 - J_3$

		Trabalhos		
		J_2	J_1	J_3
	M_1	83	137	152
	M_2	86	216	227
	M_3	175	232	281
	M_4	233	299	330
		C_{max} = 330		

(d) $J_2 - J_3 - J_1$

		Trabalhos		
		J_2	J_3	J_1
Máquinas	M_1	83	98	152
	M_2	86	109	231
	M_3	175	224	247
	M_4	233	264	330
		C_{max} = 330		

(e) $J_3 - J_1 - J_2$

		Trabalhos		
		J_3	J_1	J_2
	M_1	15	69	152
	M_2	26	148	155
	M_3	75	164	253
	M_4	106	230	311
		C_{max} = 311		

(f) $J_3 - J_2 - J_1$

		Trabalhos		
		J_3	J_2	J_1
	M_1	15	98	152
	M_2	26	101	231
	M_3	75	190	247
	M_4	106	248	314
		C_{max} = 314		

Enscore e Ham (1983) e o *Iterated Local Search* descrito em Stützle (1998) foram considerados os mais eficientes até o ano de 2004.

De 2004 até 2016 mais de 100 novos algoritmos foram propostos na literatura de acordo com Fernandez-Viagas, Ruiz e Framinan (2016). Alguns desses métodos melhoraram os algoritmos existentes, como por exemplo o Iteração Gulosa [28] (IG) proposto por Ruiz e Stützle (2007).

Uma nova tendência vem surgindo na área de otimização combinatória: a junção de métodos exatos, heurísticas e meta-heurísticas para gerar os chamados algoritmos híbridos. Juntamente com os algoritmos híbridos, os algoritmos baseados em fenômenos naturais também obtiveram uma crescente muito forte na última década. Yenisey e Yagmahan (2014) citam alguns exemplos: Evolução Diferencial [29], Algoritmo do Canguru [30], Colônia de Abelhas Artificiais, Otimização em Colônia de Formigas, Eletromagnetismo como Algoritmo [31],

[28] Tradução direta do inglês de *Iterated Greedy*
[29] Tradução direta do inglês de *Differential Evolution*
[30] Tradução direta do inglês de *Kangaroo Algorithm*
[31] Tradução direta do inglês de *Electromagnetism-like Algorithm*

Algoritmo de Colônia Artificial de Ervas Daninhas [32], Sistema Imunológico Artificial, Algoritmo Enxame Artificial de Peixes [33].

No entanto, Fernandez-Viagas, Ruiz e Framinan (2016) mostram que o estado da arte do PFSP é um tanto quanto obscuro, uma vez que não existe homogeneidade no momento de comparar os algoritmos. Alguns fatores que causam são:

- Diferentes condições para comparar os algoritmos:
  - Testado em diferentes condições de computador (diferentes linguagens de programação e/ou computadores, sistemas operacionais, etc.);
  - Comparação de algoritmos com diferentes usos de tempo de CPU.
- Muitos algoritmos são comparados de forma não conclusiva:
  - Falta de comparação com o estado da arte;
  - Escolha das melhores execuções realizadas em cada instância para aumentar o impacto dos resultados, em vez de escolher média dos resultados.
- Novos avanços na avaliação dos algoritmos:
  - Uma referência mais extensa de instâncias foi recentemente proposta por Vallada, Ruiz e Framinan (2015). Este teste pode ser usado para estabelecer diferenças estatísticas entre os algoritmos de uma forma sólida, diferentemente do que pode ser feito com *benchmarks* mais antigos.

2.2.4 Complexidade

Foi provado por Garey, Johnson e Sethi (1976) que o PFS que tem como medida de desempenho minimizar a duração total da programação (*makespan*) com três máquinas é fortemente NP-difícil. Portanto podemos concluir que para um número maior de máquinas ($m > 3$), o PFSP continuará sendo NP-difícil. Com menos que três máquinas o problema tem complexidade $O(n \log n)$.

Conforme visto na Figura 1c, que contém uma hierarquia da complexidade das medidas de desempenho na qual o *makespan* (C_{max}) está entra as medidas de menor complexidade, podemos afirmar que para qualquer outra medida de desempenho, o PFSP manterá a característica de NP-difícil, visto que com a medida de desempenho *makespan* esta característica é também observada.

Com base nessas afirmações podemos concluir que o PFSP ($Fm \mid prmu \mid C_{max}$) está na classe dos problemas NP-difíceis.

[32] Tradução direta do inglês de *Artificial Weed Colony Algorithm*
[33] Tradução direta do inglês de *Artificial Fish Swarm Algorithm*

3 Revisão Sistemática da Literatura

Revisão Sistemática da Literatura (RSL) é um método científico muito bem definido, com a finalidade de identificar, analisar e interpretar todas as evidencias disponíveis relacionadas a uma certa área da ciência. O uso de um método sistemático amplifica a confiabilidade e a precisão nos resultados e conclusões de uma pesquisa. De acordo com Mulrow (1994), mesmo a revisão sistemática consumindo muito tempo e recursos, ainda assim é mais rápido e menos custoso realizar uma RSL do que iniciar um novo estudo completo em uma área que já possui resultados publicados, mas que ainda não foram explorados da forma correta.

O estudo de Mulrow (1994) apresenta nove motivos para a realização da RSL:

1. Refinamento das quantidades incontroláveis de informação;
2. Integração de informações críticas para tomar decisões;
3. É uma eficiente técnica de pesquisa científica;
4. É preciso menos pesquisas para estabelecer a generalização dos resultados científicos;
5. Pode determinar a consistência entre os estudos de iguais opiniões ou até mesmo entre os estudos de diferentes opiniões;
6. Explicar inconsistências e conflitos de dados;
7. O uso de análises estatísticas se torna uma ferramenta poderosa, com a qual os pesquisadores podem superar o corpo de evidencias, examinar o atual cenário e traçar novas direções;
8. Permite maior precisão nas estimativas de risco ou tamanho do efeito do estudo;
9. É precisa, ou pelo menos melhora a reflexão da realidade.

Uma das principais vantagens da RSL em relação a revisão não sistemática é a sua replicabilidade, pois essa segue um protocolo de pesquisa muito bem elaborado e descrito, proporcionando a qualquer pessoa a repetição da revisão feita neste trabalho. Outra vantagem é a qualidade obtida nos resultados de pesquisa, pois as fontes e as estrategias de busca, bem como as questões de avaliação, são definidas antes da realização da pesquisa em si, isto faz com que os estudos mais relevantes não sejam eliminados quando comparados com aqueles que não devem ser lidos.

3.1 Protocolo de pesquisa

O protocolo descreve todo o planejamento da pesquisa. Trata-se do primeiro passo para a garantia da qualidade do trabalho científico. O protocolo deste trabalho será descrito a seguir.

3.1.1 Bases de Busca

Uma base de busca indica um site ou repositório em que será feita uma pesquisa para encontrar artigos, monografias, dissertações e teses. A seguir estão listadas as bases pesquisadas neste trabalho.

- Springer - <www.springerlink.com>;
- IEEE Xplore- <www.ieeexplore.ieee.org>;
- ScienceDirect - <www.sciencedirect.com>;
- ACM - <https://dl.acm.org/advsearch.cfm>;
- BDTD - <http://bdtd.ibict.br>.

3.1.2 Palavras chave

Para se fazer uma busca deve-se escolher quais são as palavras fundamentais e suas variantes a serem procuradas. Nesta monografia as seguintes palavras foram tidas como as mais importantes.

- Flow-Shop;
- Flow Shop;
- FlowShop.

A palavra 'permutacional' não foi utilizada com as outras (por exemplo: 'Flow Shop Permutacional') pelo fato de alguns autores não a utilizarem em sua terminologia mesmo a implementando de fato. Caso a palavra fosse adicionada, o filtro da busca ficaria incorreto e a pesquisa devolveria menos trabalhos pelo fato de excluir os que não a utilizam.

3.1.3 String de busca

A *string* de busca representa a maneira com que as palavras chaves serão condicionadas e organizadas afim de se realizar a pesquisa em uma base de busca.

A *string* apresentada a seguir é genérica, pois cada base de busca trata a *string* de um maneira diferente. As strings utilizadas em cada base estão descritas na seção 3.1.4.

- "(flow shop) OU (flowshop) OU (flow-shop)".

3.1.4 Refinamento da String de Busca

Para cada base de dados acima há uma maneira de se pesquisar. Por este motivo a *string* de busca deve ser modificada em cada uma destas bases a fim de obter um maior desempenho e otimização.

3.1.4.1 Springer, IEEE Xplore, ScienceDirect, BDTD

- “flow shop” OR “flowshop” OR “flow-shop”.

3.1.4.2 ACM

- acmdlTitle:(“flow shop” or “flowshop” or “flow-shop”) OR recordAbstract:(“flow shop” or “flowshop” or “flow-shop”).

3.1.5 Filtros da busca

O filtro serve para delimitar a busca de acordo com algumas restrições com a finalidade da pesquisa não ser genérica.

- Executar a *string* de busca em cada base do ano de 2013 até 2017;
- Buscas as palavras chaves somente no título e no resumo dos trabalhos;
- Idiomas: Inglês, português e espanhol.

3.1.6 Dados da busca

Os dados de busca de cada base de pesquisa serão salvos em documentos específicos para que pesquisas futuras possam repetir os experimentos e avaliar os resultados deste trabalho de maneira fiel. Para cada base de busca serão armazenados:

  - a *string* de busca utilizada;
  - a data de execução;
  - quantos artigos foram encontrados;
  - observações (caso existam).

3.1.7 Ordenamento das publicações

As publicações serão ranqueadas pelo número de citações, pela recência e pelo fator de impacto da revista, jornal ou congresso em que o trabalho foi publicado.

3.1.8 Questões de avaliação de qualidade

As questões de avaliação têm a finalidade de extrair informações relevantes dos trabalhos encontrados para que seja feita uma classificação destes. As perguntas devem ser respondidas realizando uma leitura superficial no trabalho.

Q1. O trabalho aborda o problema *flow shop* permutacional?

Q2. O autor utiliza a revisão sistemática da literatura?

Q3. O *makespan* é utilizado como função objetivo?

Q4. Os códigos fontes estão publicamente disponíveis?

Q5. Os pseudocódigos estão claramente descritos no trabalho?

Q6. O *benchmark* utilizado está disponível publicamente?

Q7. Os resultados obtidos são disponibilizados?

3.1.9 Critérios de inclusão e exclusão

Tabela 4 – Pesos das questões de avaliação utilizadas na Revisão Sistemática da Literatura

Peso	**Questão**
3	Q1
2	Q3, Q6
1	Q5, Q7, Q2, Q4

Cada questão descrita na subseção 3.1.8 terá um peso, como pode-se ver na Tabela 4. Uma nota será atribuída a cada questão conforme ela for respondida, onde a nota pode ser 10 (caso a resposta seja satisfatória), 5 (parcialmente satisfatória) ou 0 (nada satisfatória). Ao final da leitura do artigo será feita uma soma ponderada para saber quantos pontos o trabalho atingiu seguindo a seguinte formula:

$$n = \sum_{k=1}^{7} P_k \cdot Q_k \quad \text{com, P = peso e Q = questão} \tag{3.1}$$

O valor mínimo para que um trabalho seja colocado na lista de incluídos para leitura profunda é 90 pontos. Para ficar na lista dos parcialmente excluídos deve-se obter no mínimo 70 pontos, onde os trabalhos nesta lista podem ser lidos ou não, dependendo do nível de satisfação encontrado na lista dos incluídos para a leitura profunda. Por fim, qualquer pontuação abaixo de 70 colocará o trabalho na lista dos totalmente excluídos, onde estes nunca serão lidos de maneira profunda em hipótese alguma. Na Tabela 5 estão apresentados os critérios para a inclusão ou exclusão de um trabalho, onde n é a nota do trabalho.

Tabela 5 – Limites para inclusão e exclusão das publicações

Situação	**Nota**
Incluído	$n >= 90$
Parcialmente incluído	$90 < n <= 70$
Totalmente excluído	$n < 70$

3.2 Resultados da Pesquisa

Nessa seção serão apresentados os resultados da pesquisa seguindo o protocolo descrito na seção 3.1.

3.2.1 Busca Prévia

Foi realizada uma busca prévia nas bases descritas na subseção 3.1.1 com a finalidade de encontrar os artigos para leitura. No total foram encontrados 29944 artigos, e destes, 114 foram considerados para uma leitura panorâmica. Alguns dados serão mostrados a seguir:

3.2.1.1 ACM

- Data da busca: 21/03/2017
- Quantidade de artigos encontrados: 29842
  - Observação: foram considerados apenas os 10 primeiros artigos, ordenando a busca por relevância.

3.2.1.2 BDTD

- Data da busca: 22/03/2017
- Quantidade de artigos encontrados: 18

3.2.1.3 IEEE Xplore

- Data da busca: 20/03/2017
- Quantidade de artigos encontrados: 10
  - Observação: Foram considerados apenas os artigos de acesso liberado.

3.2.1.4 ScienceDirect

- Data da busca: 20/03/2017
- Quantidade de artigos encontrados: 33

3.2.1.5 Springer

- Data da busca: 19/03/2017
- Quantidade de artigos encontrados: 43
  - Observação: Foram considerados apenas os artigos de acesso liberado.

3.2.2 Leitura panorâmica

Foi realizada uma leitura panorâmica nos 114 trabalhos encontrados na subseção 3.2.1 levando em consideração apenas o título, o resumo e a conclusão dos trabalhos. A partir desta leitura, detectou-se que 38 trabalhos realmente relacionavam-se com o PFSP e estes foram então escolhidos para uma leitura intermediária.

3.2.3 Leitura intermediária

O objetivo da leitura intermediária foi responder as questões de avaliação de qualidade presentes na subseção 3.1.8. Estas respostas estão presentes na Tabela 6.

É importante observar que em nenhum trabalho foi realizada a revisão sistemática da literatura e que apenas um autor disponibilizou o código fonte do seu trabalho. Estes são aspectos que dificultam a repetibilidade e consequentemente dificultam o avanço científico na área do PFSP como está descrito na seção 1.5.

Conforme a Tabela 6 mostra, as questões de avaliação foram respondidas em cada publicação e receberam uma nota total. A partir destas respostas e com base nos critérios de inclusão e exclusão definidos na Tabela 5, as publicações estão com a coluna 'Total' marcada com uma cor, onde as publicações com a cor vermelha foram totalmente excluídas, já as amarelas ficaram na lista das parcialmente incluídas e as verdes foram as incluídas para

a leitura profunda. Com isto, tem-se que os trabalhos de Martin et al. (2016), o de Dubois-Lacoste, Pagnozzi e Stützle (2017) e o de Marichelvam, Tosun e Geetha (2017) foram os escolhidos para uma leitura profunda presente na subseção 3.2.4.

Tabela 6 – Notas atribuídas as publicações escolhidas para a leitura intermediária a partir das resposta as questões de avaliação de qualidade.

Publicação	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Total**
Chakroun e Melab (2013)	5	0	10	0	5	10	0	60
Jarosław, Czesław e Dominik (2013)	5	0	5	0	0	10	0	45
Jolai et al. (2013)	5	0	5	0	10	0	10	45
Lugo (2013)	5	0	10	0	10	0	10	55
Moritz et al. (2013)	10	0	5	0	10	0	10	60
Balaji e Porselvi (2014)	5	0	10	0	10	10	10	75
Bożejko et al. (2014)	5	0	0	0	0	10	10	45
Campos e Arroyo (2014)	5	0	5	0	0	0	10	35
Ibrahem, Elmekkawy e Peng (2014)	5	0	0	0	10	0	0	25
Mirabi, Ghomi e Jolai (2014)	10	0	0	0	5	0	10	45
Pugazhenthi e Xavior (2014)	10	0	10	0	5	10	10	85
Sioud, Gagné e Gravel (2014)	10	0	10	0	0	10	10	80
Villadiego (2014)	10	0	0	0	10	10	10	70
Campos (2015)	10	0	0	0	10	0	10	50
Carniel (2015)	5	0	10	0	10	10	10	75
Holanda (2015)	10	0	10	0	0	10	10	80
Keller, Schönborn e Reinhart (2015)	10	0	0	0	0	0	10	40
Komesu (2015)	10	0	0	0	5	10	10	65
Li, Dai e Zhang (2015)	10	0	10	0	5	5	10	75
Li et al. (2015)	10	0	0	0	10	0	10	50
Moritz et al. (2015)	5	0	5	0	0	10	10	55
Nouri e Ladhari (2015)	10	0	10	0	10	0	5	65
Rojas (2015)	5	0	10	0	10	10	5	70
Sadaqa e Moraga (2015)	5	0	10	0	5	10	5	65
Yahyaoui et al. (2015)	10	0	10	0	0	10	5	75
Aurich et al. (2016)	5	0	10	0	0	0	10	45
Eddaly, Jarboui e Siarry (2016)	5	0	10	0	10	10	10	75
Ferone et al. (2016)	10	0	10	0	0	5	10	70
Kieskoski (2016)	10	0	10	0	10	0	10	70
Leeftink et al. (2016)	5	0	0	0	0	0	10	25
Li, Pan e Duan (2016)	5	0	0	0	10	0	10	35
Li e Ma (2016)	10	0	5	0	10	10	10	80
Mansouri, Aktas e Besikci (2016)	10	0	5	0	10	5	10	70
Martin et al. (2016)	10	0	10	10	0	10	10	90
Simões (2016)	10	0	5	0	10	5	10	70
Yang, Ma e Wu (2016)	5	0	0	0	0	5	0	25
Dubois-Lacoste, Pagnozzi e Stützle (2017)	10	0	10	0	10	10	10	90
Marichelvam, Tosun e Geetha (2017)	10	0	10	0	10	10	10	90

3.2.3.1 Publicações levantadas

Chakroun e Melab (2013) investigam duas abordagens diferentes que projetam algoritmos B&B irregulares em cima de GPUs para o PFS. A primeira é uma extensão de outro modelo e consiste em explorar em paralelo diferentes subespaços da árvore. A segunda abordagem proposta consiste em transformar a busca de árvore irregular em tarefas regulares (operações) a serem realizadas em paralelo. Usando estruturas de dados persistentes, os diferentes operadores são aplicados em paralelo em um *pool* de nós pendentes.

Jarosław, Czesław e Dominik (2013) propõem uma nova ideia do uso do método de recozimento simulado para resolver o problema *flow shop* multiobjetivo. Uma função para medir a diferença escalar entre duas soluções multiobjetivos foi desenvolvida.

O artigo de Jolai et al. (2013) foca em resolver o problema bi-objetivo para o problema *flow shop* híbrido (FSH) de dois estágios sem espera. Os objetivos considerados neste estudo são a minimização do *makespan* e o atraso máximo de trabalhos. São desenvolvidos três métodos de otimização bi-objetivos baseados em recozimento simulado, chamados recozimento simulado ponderado clássico (RSPC), recozimento simulado ponderado normalizado (RSPN) e recozimento simulado difuso (RSD) para resolver o problema com o objetivo de encontrar aproximações do ótimo de Pareto.

Já Lugo (2013) estuda o FSH com *buffer* limitado e a sua medida de desempenho é o *makespan*. Aqui foi proposto um modelo de Programação Inteira Mista (PIM) para resolver instâncias de até 5 estações e 9 trabalhos. Para as instancias maiores foram propostos algoritmos baseados nas heurísticas NEH e CDS e na meta-heurística BLI.

Dois novos métodos de classificação chamados de relação WL (*won and lost*) e relação *Point* combinados com o Otimização em Colônia de Formigas (OCF) baseado em população (P-OCF) são propostos por Moritz et al. (2013) para resolver o PFS multi-objetivo, com as medidas de desempenho tempo total, tempo de fluxo, tempo total dos trabalhos ociosos e tempo total das máquinas ociosas.

O artigo de Balaji e Porselvi (2014) aborda o problema de escalonamento de lotes no Sistema de Manufatura Flexível Multicelular, tendo um tempo de configuração de lote dependente da sequência com características de *flow shop*. A medida de desempenho é o *makespan*. Foi desenvolvido um algoritmo de Sistemas Imunológicos Artificiais (SIA) e um algoritmo de Recozimento Simulado.

Bożejko et al. (2014) trabalha com o problema FSH com configurações da máquina e a medida de desempenho é a minimização do tempo de ciclo. O algoritmo aqui implementado é o Busca Tabu.

Campos e Arroyo (2014) propõem um Algoritmo Genético de Ordenação Elitista Não-Dominante conhecido como *(NSGA2)* acoplado à estratégia Iteração Gulosa. O trabalho aqui estudado é bi-objetivo minimizando o tempo total do fluxo e o atraso total tratando o pro-

blema *flow shop* de montagem em três estágios.

O artigo de Ibrahem, Elmekkawy e Peng (2014) considera um problema *flow shop* celular com sequência familiar de tempo de configuração para minimizar o tempo total do fluxo. Dois algoritmos meta heurísticos baseados em AG e Otimização de Enxame de Partículas (OEP) são propostos para resolver o problema.

Na pesquisa de Mirabi, Ghomi e Jolai (2014) se estuda o PFSP. O tempo de preparação e o custo de configuração são dependentes da sequência e a medida de desempenho é constituída de três critérios: atraso, retenção e custo de configuração. Um novo Algoritmo Genético Híbrido (AGH) com três operadores genéticos é desenvolvido.

No trabalho de Pugazhenthi e Xavior (2014) foi analisado o PFSP com objetivo de minimizar o *makespan* utilizando uma nova heurística chamada BAT juntamente com AG. Para chegar no *makespan* mínimo, um método de engenharia reversa foi realizado partindo do limite inferior (*lower bound*) proposto por Taillard (1993).

O problema abordado por Sioud, Gagné e Gravel (2014) é o FSH com tempos de configuração dependentes da sequência e a função objetivo é minimizar o *makespan*. Foi proposta um OCF com uma nova heurística construtiva usada na melhoria local.

Villadiego (2014) aborda o problema de dimensionamento e sequenciamento de lotes no *flow shop*. As máquinas possuem capacidades diferentes de produção e tempos de preparação que dependem da sequência. Foram propostos três algoritmos heurísticos baseados na IG. O primeiro é o IG+HR, é uma adaptação da heurística Horizonte Rolante. O segundo é o IG+MDL, onde foi utilizado um método de Melhoria de Dimensionamento de Lotes. Já o terceiro algoritmo, denominado IG+F&O, é uma adaptação da heurística *Fix and Otimize*.

Campos (2015) trata do problema *assembly flow shop* de três estágios. Para a abordagem com uma função objetivo são propostos quatro algoritmos baseados em GRASP-RVND, BLI, IG, BLI-IG. Já na multi-objetivo são propostos dois algoritmos com base no NSGA2-IG

Carniel (2015) fazem um estudo de caso sobre a inclusão de pessoas com deficiência, um e dois trabalhadores, no PFS objetivando minimizar o *makespan*. Foram desenvolvidas duas heurísticas baseadas no IG e um algoritmo de Programação Dinâmica. Foi concluído que a inserção de pessoas com deficiências no PFS é viável economicamente.

Holanda (2015) mostra que o Problema do Caixeiro Viajante (PCV) pode resolver instâncias do PFS, utilizando a Matriz Distância 'D' de um PCV para o FSP. Para isto foi proposto um AG tendo como função objetivo o *makespan*.

Keller, Schönborn e Reinhart (2015) propõem uma maneira de resolver o problema de capacidades energéticas através do FSH. Um algoritmo de RS foi utilizado para resolver este problema com as medidas de desempenho data de vencimento e custo da energia.

Komesu (2015) apresentam a heurística Busca em Cluster Evolucionária (BCE) para a resolução do PFS com *buffer* zero. A medida de desempenho aqui utilizada é o atraso total.

Foi proposto por Li, Dai e Zhang (2015) uma nova função objetivo para o PFS que leva em consideração a minimização do *makespan* e do tempo médio de processamento. Também foi apresentada uma nova heurística chamada Espaço de Estado com Cabeça, Corpo e Cauda.

Li et al. (2015) apresentam Métodos de Escalonamento de Trajetória (MET) para o PFSP com o objetivo de minimizar o atraso total. Um total de 18 METs são construídos por meio de diferentes combinações de composições heurísticas.

Moritz et al. (2015) desenvolvem dois métodos para classificação de soluções de problemas de otimização multi-objetivo chamados de Relação Ganha-Perde (RGP) e Relação de Pontos (RP). Os métodos podem ser usados para selecionar soluções de um conjunto de soluções não determinadas.

Nouri e Ladhari (2015) Este trabalho apresenta e avalia meta heurísticas para o PFSP de bloqueio sujeito a objetivos regulares, onde as meta-heurísticas apresentadas são o AG e o algoritmo de Colônia de Abelhas Artificiais (CAA). As medidas de desempenho utilizadas são o *makespan*, tempo total de fluxo e o atraso total.

Na tese de Rojas (2015) são propostos métodos para resolver o PFS não-permutacional, usando o mesmo tempo e esforço que os métodos do estado da arte usam para o PFSP. Também são propostos métodos para resolver o problema combinado de designação de trabalhadores heterogêneos e escalonamento de tarefas em um *flow shop*.

A pesquisa de Sadaqa e Moraga (2015) se concentra em encontrar uma solução para o *flow shop* com restrição de bloqueio de mais de duas máquinas com objetivo de minimizar o *makespan*. A solução proposta nesta pesquisa inclui o uso de uma meta heurística recém desenvolvida conhecida como Meta-heurística para Busca Aleatória de Prioridade (Meta-BAP).

No artigo de Yahyaoui et al. (2015), uma Busca Local Iterada (BLI) é combinada com uma hiper-heurística de Descida Variável na Vizinhança (DVV). A hiper-heurística proposta combina heurísticas de baixo nível. Diversas variantes da literatura dentro do BLI proposto foram implementadas com os objetivos de minimizar o *makespan* e o tempo total de fluxo dos trabalhos.

Aurich et al. (2016) descreve uma solução para um problema de um conjunto de placas de circuito impresso em um ambiente de programação FSH. O objetivo é minimizar o *makespan* e o atraso total. O artigo compara três abordagens: um algoritmo de Otimização Integrada Baseada em Simulação (OIBS) desenvolvido pelos autores e as meta-heurísticas RS e BT.

Eddaly, Jarboui e Siarry (2016) abordam o problema de escalonamento de *flow shop* com restrições de bloqueio. O objetivo é minimizar o *makespan*. Foram propostos dois algoritmos para a resolução do problema, onde um deles é o algoritmo combinatório de OEP e o outro é um algoritmo híbrido que mistura o BLI com o combinatório de OEP.

No artigo de Ferone et al. (2016), foi apresentado um algoritmo sim-heurístico, isto é, combinação de um algoritmo de simulação com uma meta-heurística que integra a simulação de Monte Carlo (MC) em uma estrutura GRASP para resolver o PFSP com tempos de processamento estocásticos, onde a função objetivo é o *makespan*.

Kieskoski (2016) propõe um algoritmo baseado na meta-heurística OCF para o PFSP com sequência dependente dos tempos de configuração. A medida de desempenho utilizada para avaliar a solução é o *makespan*.

Com o objetivo de reduzir a carga de trabalho e o tempo de resposta em um laboratório de histopatologia, Leeftink et al. (2016) desenvolve uma abordagem composta por duas fases para resolver o problema. O ambiente é o FSH com lotes paralelos. Primeiramente o autor resolve o problema de lotes usando o PLIM utilizando como função objetivo a minimização do nível de estoque máximo. Na segunda etapa resolve-se o problema de escalonamento utilizando uma lista de algoritmos multi-fase com o objetivo de minimizar o atraso total.

Li, Pan e Duan (2016) propõe um algoritmo melhorado de CAA discretas para resolver o problema FSH com características de saltos de operação dinâmica em sistemas de ferro fundido. a medida de desempenho utilizada é a data de vencimento.

No artigo Li e Ma (2016), um novo algoritmo de Busca Memética (BM) multi-objetivo é proposto para resolver o PFSP tendo o *makespan* e o tempo total de fluxo como sendo as medidas de desempenho.

Mansouri, Aktas e Besikci (2016) cria uma literatura chamada de *escalonamento verde* para o PFSP de duas máquinas dependentes da sequência com as preocupações do nível de serviço e de consumo de energia que limitam a fabricação sustentável e a tomada de decisões multicritério. As medidas de desempenho utilizadas são o *makespan* e o consumo de energia. Os algoritmos implementados são o PLIM e uma heurística construtiva que inclui técnicas de um cronograma de desenvolvimento heurístico e também uma busca local.

Martin et al. (2016) propõem uma estrutura geral distribuída baseada em agentes, onde cada agente implementa uma combinação diferente de busca local/meta-heurística. Dois domínios de problema foram atacados, o PFSP e o problema de Roteamento de Veículos Capacitados (PRVC). Para o PFSP o objetivo foi minimizar o *makespan*.

Simões (2016) desenvolvem uma ferramenta computacional que utiliza um algoritmo híbrido que combina as meta-heurísticas BT e AG, de forma a resolver o PFSP com tempos de configuração. A funao objetivo é minimizar o *makespan* aplicado à montagem de

placas eletrônicas em ambientes *High-Mix* e *Low-Volume*.

O artigo de Yang, Ma e Wu (2016) propõe a resolução do *flow shop* com múltiplas linhas de produção de estruturas de concreto pré-moldadas e desenvolve uma abordagem de otimização correspondente para facilitar o planejamento otimizado usando AG.

3.2.3.2 Métodos e *benchmarks*

Os métodos/algoritmos e os *benchmarks* utilizados pelos autores em cada trabalho foram encontrados a partir da leitura realizada nesta subseção e estão dispostos na Tabela 7.

Tabela 7 – Algoritmos e *Benchmarks* analisados na RSL

Publicação	**Método Utilizado**	***Benchmark***
Chakroun e Melab (2013)	Branch&Bound	Taillard (1993)
Jarosław, Czesław e Dominik (2013)	RS	Taillard (1993), Minella, Ruiz e Ciavotta (2008)
Jolai et al. (2013)	RS	Gerados pelo autor
Lugo (2013)	PIM, CDS, NEH, BLI	Gerados pelo autor
Moritz et al. (2013)	OCF	Outros autores
Balaji e Porselvi (2014)	SIA, RS	Das e Canel (2005)
Bożejko et al. (2014)	BT	Ruiz, Şerifoğlu e Urlings (2008)
Campos e Arroyo (2014)	NSGA2+IG	Instâncias Aleatórias
Ibrahem, Elmekkawy e Peng (2014)	OEP, AG	Instâncias Aleatórias
Mirabi, Ghomi e Jolai (2014)	AGH	Instâncias Aleatórias
Pugazhenthi e Xavior (2014)	AG+BAT	Taillard (1993)
Sioud, Gagné e Gravel (2014)	OCF	<http://soa.iti.es/problem-instances>
Villadiego (2014)	IG	Mohammadi et al. (2010)
Campos (2015)	GRASP+RVND, BLI, IG, BLI+IG e NSGA2+IG	Instâncias Aleatórias
Carniel (2015)	IG e PD	Taillard (1993), Carlier (1978)
Holanda (2015)	AG	Taillard (1993), Beasley (1990)
Keller, Schönborn e Reinhart (2015)	RS	Instâncias Aleatórias
Komesu (2015)	BCE	Taillard (1993)

Li, Dai e Zhang (2015)	SS+HBT	Gerados pelo autor, Taillard (1993)
Li et al. (2015)	MET	Gerados pelo autor usando Taillard (1993)
Moritz et al. (2015)	RGP e RP	Taillard (1993), Reeves (1995), Carlier (1978), Heller (1960)
Nouri e Ladhari (2015)	AG e CAA	Taillard (1993), Instâncias Aleatórias
Rojas (2015)	IG, BLI, NEH, NEH_{BR} e Busca de Dispersão	Taillard (1993), Demirkol, Mehta e Uzsoy (1998)
Sadaqa e Moraga (2015)	Meta-BAP	Taillard (1993)
Yahyaoui et al. (2015)	BLI+DVV	Taillard (1993)
Aurich et al. (2016)	OIBS, RS e Busca Tabu	Mundo real
Eddaly, Jarboui e Siarry (2016)	CPSO e CPSO+BLI	Taillard (1993), Mundo real
Ferone et al. (2016)	MC+GRASP	Taillard (1993)
Kieskoski (2016)	OCF	Instâncias Aleatórias
Leeftink et al. (2016)	PLIM+Algoritmos Multi-fase	Instâncias Aleatórias
Li, Pan e Duan (2016)	CAA	Mundo real
Li e Ma (2016)	BM	Taillard (1993)
Mansouri, Aktas e Besikci (2016)	PLIM e Heurística Construtiva	Taillard (1990), Ruiz, Maroto e Alcaraz (2005), Naderi, Zandieh e Roshanaei (2009), Ruiz e Stützle (2008)
Martin et al. (2016)	Busca Local+NEH Aleatório	Taillard (1993)
Simões (2016)	AG+BT	Mundo real, TSPLIB-<http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/atsp/>
Yang, Ma e Wu (2016)	AG	Gerados pelo autor
Dubois-Lacoste, Pagnozzi e Stützle (2017)	IG_{BLSP}	Taillard (1993)
Marichelvam, Tosun e Geetha (2017)	BMH	Taillard (1993)

3.2.4 Leitura profunda

De acordo com a leitura intermediária realizada na subseção 3.2.3 três publicações foram escolhidas para uma leitura mais profunda e minuciosa com a finalidade de se escolher dois trabalhos para compará-los nesta monografia. Os três apresentaram a mesma pontuação final na Tabela 6 e se mostraram equivalentes neste aspecto.

A partir da leitura proposta nesta subseção as publicações de Dubois-Lacoste, Pagnozzi e Stützle (2017) e a de Marichelvam, Tosun e Geetha (2017) foram escolhidas para serem implementadas e comparadas entre si pelo fato de apresentarem resultados bons e por ter sido testadas em todas as 120 instâncias propostas por Taillard (1993). Outro fator importante na escolha destes trabalhos foi a de que os autores do Dubois-Lacoste, Pagnozzi e Stützle (2017) relatam que o algoritmo proposto por eles é uma melhoria do já consagrado trabalho realizado por Ruiz e Stützle (2007). Por outro lado os autores do artigo Marichelvam, Tosun e Geetha (2017) dizem que o algoritmo proposto por eles é mais eficiente e efetivo do que muitos dos métodos existentes na literatura.

O trabalho proposto por Martin et al. (2016) propõe uma estrutura geral distribuída baseada em agentes, onde cada agente implementa uma combinação diferente de busca local/meta-heurística. Dois domínios de problema foram atacados, o PFSP e o Problema de Roteamento de Veículos Capacitados (PRVC). Este trabalho também apresentou bons resultados e o *benchmark* escolhido para testes também foi o de Taillard (1993), no entanto a carga de testes realizadas teve como base apenas 12 instâncias, cerca de 10% dos testes realizados nos outros dois trabalhos, este foi um dos motivos da não escolha deste trabalho para implementação. Outro motivo pela não escolha foi que implementação não teve foco total no PFSP uma vez que o autor também implementou um algoritmo para o PRVC.

4 Algoritmos Estudados

Neste capítulo estão descritos os algoritmos IG_{BLSP} e BMH, que foram selecionados conforme está relatado na subseção 3.2.4 para serem estudados e implementados nesta monografia. Também foi mostrado o passo a passo da implementação da heurística NEH, utilizada para construir a solução inicial do IG_{BLSP} e do BMH. Todos os algoritmos estão detalhados com os seus respectivos pseudocódigos, uma explicação de todos os passos e parâmetros para a implementação e também um exemplo numérico para um melhor entendimento do funcionamento de cada um. Os códigos fonte de cada um dos três algoritmos implementados neste trabalho estão disponíveis publicamente e podem ser encontrados em <https://github.com/ViniciusTxr/algoritmos-flowshop> e nos Apêndices B, C e D.

4.1 Heurística NEH

A heurística Nawaz, Enscore and Ham (NEH) apresentada em Nawaz, Enscore e Ham (1983), é uma heurística construtiva simples e de alto desempenho com a finalidade de minimizar o *makespan* para o PFSP.

Como visto na seção 2.2, o PFSP é um problema de busca combinatorial com *n!* possíveis sequências candidatas a solução. No entanto, o NEH consegue encontrar uma solução aceitável com $\left(\frac{n \cdot (n+1)}{2} - 1\right)$ sequências encontradas, onde destas sequências, n são completas e o restante são sequências parciais.

4.1.1 Descrição do algoritmo

1. Para cada trabalho j, calcule o tempo total de processamento P_j nas m máquinas:

$$P_j = \sum_{i=1}^{m} p_{ij} \qquad \forall j, j \in N$$

   onde p_{ij} representa o tempo de processamento do trabalho j na máquina i.

2. Ordene de forma decrescente os trabalhos de acordo com P_j. Coloque o resultado da ordenação em β.

3. Pegue os dois primeiros trabalhos de β, ou seja, β_1 e β_2 e encontre a melhor sequência (menor C_{max}) entre os dois, calculando o *makespan* para as duas sequências possíveis (β_1-β_2 ou β_2-β_1). Coloque a melhor sequência de trabalhos em ϕ. Depois de encontrar ϕ, as posições dos dois trabalhos não devem mais ser trocadas entre si nos próximos passos. Faça $i \leftarrow 3$.

4. Pegue o i-ésimo trabalho da lista β, ou seja, β_i. Encontre a melhor sequência colocando β_i em todas as i posições possíveis de ϕ, note que haverá i permutações neste passo. Atualize ϕ com a nova melhor sequência encontrada. Assim como no passo anterior, as posições dos trabalhos de ϕ não devem mais ser trocadas entre si daqui para frente.

5. Se $i = n$, PARE e devolva ϕ. Senão, faça $i \leftarrow i + 1$ e vá ao passo 4.

4.1.2 Exemplo numérico

Aqui será apresentado um exemplo numérico com o objetivo de facilitar o entendimento do algoritmo NEH.

Na Tabela 8 são apresentados os tempos de processamento p_{ij} de cada trabalho em cada máquina.

Tabela 8 – Tempos de processamento para um problema de escalonamento.

		Trabalhos				
		J_1	J_2	J_3	J_4	J_5
Máquinas	M_1	54	83	15	71	77
	M_2	79	3	11	99	56
	M_3	16	89	49	15	89
	M_4	66	58	31	68	78

Já na Tabela 9 está representada a execução do passo 1 do algoritmo NEH. O somatório dos tempos de processamento dos trabalhos foi calculado em cada máquina e o resultado pode ser visto na linha P_j. Por exemplo, para o trabalho J_1 o cálculo do somatório de P_1 se deu por 54 + 79 + 16 + 66 = 215. Para os outros trabalhos o calculo é feito de maneira semelhante.

Tabela 9 – Resolução inicial para o algoritmo NEH.

		Trabalhos				
		J_1	J_2	J_3	J_4	J_5
Máquinas	M_1	54	83	15	71	77
	M_2	79	3	11	99	56
	M_3	16	89	49	15	89
	M_4	66	58	31	68	78
	P_j	215	233	106	253	300

A partir dos valores de P_j encontrados na Tabela 9 pôde-se ser realizada a execução do passo 2. Com isso a ordenação decrescente de P_j resultou em $\beta \leftarrow (J_5 - J_4 - J_2 - J_1 - J_3)$, ou seja, $\beta_1 = J_5$, $\beta_2 = J_4$, $\beta_3 = J_2$, $\beta_4 = J_1$, $\beta_5 = J_3$.

Na Tabela 10 está demonstrado o cálculo do C_{max} realizado como indicado no passo 3. A Tabela 10a faz este cálculo para a sequência $\beta_1 - \beta_2$ enquanto a Tabela 10b o faz para $\beta_2 - \beta_1$. Como o C_{max} de $\beta_1 - \beta_2$ é menor que o de $\beta_2 - \beta_1$, então $\phi \leftarrow (J_5 - J_4)$ e a ordem dos trabalhos desta sequência não foram mais mudadas entre si daqui em diante. Foi feito $i \leftarrow 3$.

Tabela 10 – Permutações iniciais para o algoritmo NEH.

(a) $J_5 - J_4$

		Trabalhos	
		J_5	J_4
Máquinas	M_1	77	148
	M_2	133	247
	M_3	222	262
	M_4	300	368
		C_{max} = 368	

(b) $J_4 - J_5$

		Trabalhos	
		J_4	J_5
Máquinas	M_1	71	148
	M_2	170	226
	M_3	185	315
	M_4	253	393
		C_{max} = 393	

O passo 4 do algoritmo diz para inserir o i-ésimo trabalho de β em todas as posições possíveis de ϕ. Neste momento $i = 3$, então o trabalho permutado foi $\beta_3 = J_2$. Na Tabela 11 estão os resultados de C_{max} obtidos desta permutação. O menor *makespan* foi encontrado na permutação mostrada na Tabela 11c. Com isto tem-se que $\phi \leftarrow (J_5 - J_4 - J_2)$. A ordem dos trabalhos desta sequência não foram mais mudadas entre si daqui em diante.

Tabela 11 – Permutando o trabalho J_2 utilizando NEH.

(a) $J_2 - J_5 - J_4$

		Trabalhos		
		J_2	J_5	J_4
Máquinas	M_1	83	160	231
	M_2	86	216	330
	M_3	175	305	345
	M_4	233	383	451
		C_{max} = 451		

(b) $J_5 - J_2 - J_4$

		Trabalhos		
		J_5	J_2	J_4
	M_1	77	160	231
	M_2	133	163	330
	M_3	222	311	345
	M_4	300	369	437
		C_{max} = 437		

(c) $J_5 - J_4 - J_2$

		Trabalhos	
		$J_5 J_4$	J_2
	M_1	148	231
	M_2	247	250
	M_3	262	351
	M_4	368	426
		C_{max} = 426	

Seguindo para o passo 5 tem-se que $i < n$, ou seja, $3 < 5$, logo $i \leftarrow 4$. Então o passo 4 foi repetido. Agora $i = 4$, então o trabalho permutado foi $\beta_4 = J_1$. Na Tabela 12 estão os resultados de C_{max} obtidos desta permutação. Na Tabela 12a está disponibilizado o menor *makespan* dentre as permutações. Com isto obteve-se $\phi \leftarrow (J_1 - J_5 - J_4 - J_2)$. A ordem dos trabalhos desta sequência não foram mais mudadas entre si daqui em diante.

Tabela 12 – Permutando o trabalho J_1 utilizando NEH.

(a) $J_1 - J_5 - J_4 - J_2$

		Trabalhos			
		J_1	J_5	J_4	J_2
Máquinas	M_1	54	131	202	285
	M_2	133	189	301	304
	M_3	149	278	316	405
	M_4	215	356	424	482
		C_{max} = 482			

(b) $J_5 - J_1 - J_4 - J_2$

		Trabalhos			
		J_5	J_1	J_4	J_2
Máquinas	M_1	77	131	202	285
	M_2	133	212	311	314
	M_3	222	238	326	415
	M_4	300	366	434	492
		C_{max} = 492			

(c) $J_5 - J_4 - J_1 - J_2$

		Trabalhos		
		J_5J_4	J_1	J_2
Máquinas	M_1	148	202	285
	M_2	247	326	329
	M_3	262	342	431
	M_4	368	434	492
		C_{max} = 492		

(d) $J_5 - J_4 - J_2 - J_1$

		Trabalhos	
		$J_5J_4J_2$	J_1
Máquinas	M_1	231	285
	M_2	250	364
	M_3	351	380
	M_4	426	492
		C_{max} = 492	

Indo para o passo 5, como $4 < 5$, então $i \leftarrow 5$ e deve-se voltar ao 4. Tem-se que $i = 5$, então foi permutado o trabalho $\beta_5 = J_3$. Estas permutações geraram os valores de C_{max} que estão disponibilizados na Tabela 13. O menor *makespan* obtido entre as permutações está contido na Tabela 13b. A partir disso o novo valor de ϕ foi atualizado com a nova melhor sequência de trabalhos de modo que $\phi \leftarrow (J_1 - J_3 - J_5 - J_4 - J_2)$. A ordem dos trabalhos desta sequência não foram mais mudadas entre si daqui em diante.

Tabela 13 – Permutando o trabalho J_3 utilizando NEH.

(a) $J_3 - J_1 - J_5 - J_4 - J_2$

		Trabalhos				
		J_3	J_1	J_5	J_4	J_2
Máquinas	M_1	15	69	146	217	300
	M_2	26	148	204	316	319
	M_3	75	164	293	331	420
	M_4	106	230	371	439	497
		C_{max} = 497				

(b) $J_1 - J_3 - J_5 - J_4 - J_2$

		Trabalhos				
		J_1	J_3	J_5	J_4	J_2
Máquinas	M_1	54	69	146	217	300
	M_2	133	144	202	316	319
	M_3	149	198	291	331	420
	M_4	215	246	369	437	495
		C_{max} = 495				

(c) Permutação $J_1 - J_5 - J_3 - J_4 - J_2$

		Trabalhos			
		J_1J_5	J_3	J_4	J_2
Máquinas	M_1	131	146	217	300
	M_2	189	200	316	319
	M_3	278	327	342	431
	M_4	356	387	455	513
		C_{max} = 513			

(d) $J_1 - J_5 - J_4 - J_3 - J_2$

	Trabalhos			
	$J_1J_5J_4$	J_3	J_2
M_1	202	217	300
M_2	301	312	315
M_3	316	365	454
M_4	424	455	513
	C_{max} = 513		

(e) $J_1 - J_5 - J_4 - J_2 - J_3$

	Trabalhos	
	$J_1J_5J_4J_2$	J_3
M_1	285	300
M_2	304	315
M_3	405	454
M_4	482	513
	C_{max} = 513	

Avançando para o passo 5 tem-se que $i = n$, isto é, 5 = 5. Isto indica que a execução do algoritmo deve ser parada neste ponto pois o critério de parada foi atingido. A saída do algoritmo foi ϕ, a melhor solução encontrada dentre as permutações realizadas, ou seja, $\phi = (J_1 - J_3 - J_5 - J_4 - J_2)$ cujo valor de *makespan* é igual a 495.

4.2 Iteração Gulosa com Busca Local em Soluções Parciais

O algoritmo *Iteração Gulosa* com Busca Local em Soluções Parciais (IG_{BLSP}) proposto por Dubois-Lacoste, Pagnozzi e Stützle (2017) é baseado no método *Iteração Gulosa* (será chamado de IG_{RS}) desenvolvido por Ruiz e Stützle (2007) para o PFSP, e que tem como principio iterar sobre uma construção heurística por fases de destruição e reconstrução da solução. A grande diferença entre os dois modelos é que enquanto o IG_{RS} realiza uma busca local somente apos a fase de construção para melhorar a solução completa, o IG_{BLSP} também faz uma busca local depois da fase de destruição, melhorando assim a solução parcial antes da fase de construção.

Algoritmo 4.1: PSEUDOCÓDIGO IG_{BLSP}

Entrada: Solução inicial π `/* gerada pela heurística NEH */`
Saída: A melhor solução encontrada π^*

```
1  início
2      π* ← π
3      enquanto diferente do critério de parada faça
4          π_R ← Destruição(π);
5          π_R ← Busca_Local_Solução_Parcial(π_R);
6          π' ← Construção(π_R);
7          π' ← Busca_Local_Solução_Completa(π');
8          π ← Critério_Aceitação(π, π');
9          se f(π') < f(π*) então
10             π* ← π'
11         fim
12     fim
13 fim
14 retorna π*
```

Seguindo o Algoritmo 4.1 que contém o pseudocódigo IG_{BLSP}, a entrada é a solução gerada pelo algoritmo NEH desenvolvido por Nawaz, Enscore e Ham (1983) e descrito na seção 4.1. A saída é a melhor solução encontrada durante a execução do IG_{BLSP}. Já no laço principal do algoritmo, podemos ver que na linha 4 temos a fase de destruição de uma solução candidata completa. Na linha 5 será feita uma busca local na solução parcial resultante da destruição. A etapa de construção, linha 6, montará uma solução completa a partir da busca local feita anteriormente. Após isso, linha 7, novamente será realizada uma busca local, no entanto desta vez será na solução completa que resulta da construção. Finalmente, um critério de aceitação, linha 8, será aplicado para determinar em qual solução a busca continuará a ser feita e na linha 10 haverá a atualização do melhor global.

4.2.1 Fase de destruição

A fase de destruição é aplicada sobre a permutação π com n trabalhos e escolhe aleatoriamente e sem repetição K trabalhos, onde K é um parâmetro deste algoritmo e está indicado na seção 4.2.7. Esses K trabalhos serão removidos da permutação π na mesma ordem em que foram escolhidos, o resultado desse procedimento será duas subsequências. A primeira terá uma solução parcial π_R de tamanho n-K, uma vez que foram removidos K trabalhos. A segunda conterá os K trabalhos ordenados na mesma ordem em que foram removidos de π, e será chamada de π_D.

4.2.2 Busca local na solução parcial

Aqui a solução parcial π_R obtida na seção 4.2.1 será rearranjada da seguinte maneira:

1. Escolha um trabalho j de forma aleatória na permutação π_R (sem repetição).
2. Remova o trabalho j da permutação π_R.
3. Insira o trabalho j em todas as posições possíveis de $\pi_R - j$ e obtenha um conjunto de permutações.
4. Escolha a melhor permutação (menor makespan) do passo 3 e armazene em π_{aux}.
5. Se $C_{max}(\pi_{aux}) < C_{max}(\pi_R)$, então faça $\pi_R \leftarrow \pi_{aux}$ e volte ao passo 1. Senão devolva π_R.

De acordo com o passo 5, esta busca local será realizada enquanto a solução parcial estiver sendo melhorada, ou seja, até que um ótimo local seja atingido.

4.2.3 Fase de construção

A fase de construção começa com a subsequência π_R recebida da seção 4.2.2 e passara no total por k passos de reconstrução da solução, com os trabalhos de π_D sendo reinseridos em π_R. Esta reinserção começa sempre fazendo uso do primeiro trabalho de π_D, ou seja $\pi_D(1)$, e inserindo-o em todas as posições possíveis de π_R, gerando assim um conjunto de permutações. A melhor destas permutações (menor *makespan*) sera atribuída a π_R e o trabalho $\pi_D(1)$ sera removido de π_D. Caso haja empate, ou seja, mais de uma melhor solução com *makespan* igual, a que foi encontrada primeiro será escolhida. Este processo vai ser iterado ate que π_D esteja vazia. A solução final é representada por π'.

4.2.4 Busca local na solução completa

Aqui a solução completa π' encontrada na seção 4.2.3 será reconstruída assim:

1. Escolha um trabalho j de forma aleatória na permutação π'.
2. Remova o trabalho j da permutação π'.
3. Seja $i \in \{1, 2, ..., n\}$, com i começando em 1.
4. Insira o trabalho j na posição i da permutação $\pi' - j$, e armazene em π_{aux}.
5. Se $C_{max}(\pi_{aux}) < C_{max}(\pi')$, então faça $\pi' \leftarrow \pi_{aux}$ e volte ao passo 1.
6. Senão, se $i < n$, então faça $i \leftarrow i+1$ e volte ao passo 4.
7. Senão devolva π'.

A diferença desta busca para a busca da seção 4.2.2 é que esta para a busca quando acha o primeiro melhoramento, ou seja, se após o passo 4 um melhoramento for encontrado, então esta solução é aceita imediatamente e a busca recomeça ate chegar a um ótimo local.

4.2.5 Critério de aceitação

Ao final da etapa descrita na seção 4.2.4 teremos uma nova sequência π' candidata a ser uma nova solução. No entanto, esta nova sequência deve ser aceita ou não como uma solução digna a ser passada para a próxima iteração. Este processo de aceitar ou não a solução segue alguns critérios e portanto será chamado de *critério de aceitação* e tem a finalidade de evitar situações que levem a estagnação da busca devido a insuficiente diversificação da solução corrente. Por este motivo uma solução pior pode ser aceita como nova solução.

O critério de aceitação sempre aceita uma nova solução candidata π' que é de qualidade melhor ou igual quando comparada à atual. No entanto, se esta nova solução for pior, uma probabilidade será calculada para verificar a aceitação ou não da solução. Esta probabilidade de aceitar π' é dada por:

$$e^{(C_{max}(\pi)-C_{max}(\pi'))/T} \tag{4.1}$$

onde:

$$T = T_p \cdot \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}}{n \cdot m \cdot 10} \tag{4.2}$$

e T_p é um parâmetro conforme indicado na seção 4.2.7. O critério de aceitação pode ser melhor visualizado no Algoritmo 4.2.

Observação: o resultado do critério de aceitação será passado como parâmetro para a fase de destruição (seção 4.2.1) em uma próxima iteração do algoritmo caso esta ocorra.

Algoritmo 4.2: PSEUDOCÓDIGO CRITÉRIO DE ACEITAÇÃO

Entrada: π, π'
Saída: π
1 **início**
2 **se** $C_{max}(\pi') \leq C_{max}(\pi)$ **então**
3 $\pi \leftarrow \pi'$;
4 **senão se** $X < e^{(C_{max}(\pi)-C_{max}(\pi'))/T}$ **então** `/* X aleatório ∈ (0.0,1.0) */`
5 $\pi \leftarrow \pi'$;
6 **fim**
7 **fim**
8 **retorna** π

4.2.6 Atualização do melhor global

Neste ponto do algoritmo acontece ou não a atualização do melhor global encontrado. Para tal atualização, somente dois parâmetros são necessários: π^*, que é o melhor global até o momento e π', que é o valor obtido na fase de BLSC descrito na seção 4.2.4. Se $C_{max}(\pi') < C_{max}(\pi^*)$ então $\pi^* \leftarrow \pi'$, senão, o valor de π^* deve ser mantido o mesmo.

Observação: Não se deve utilizar o valor obtido no critério de aceitação (π), pois este sempre terá o *makespan* **maior ou igual** ao *makespan* encontrado na BLSC, ou seja, $C_{max}(\pi) \geq C_{max}(\pi')$.

4.2.7 Parâmetros

O número K de trabalhos a serem removidos durante a fase de destruição determina a quantidade de interrupções que serão aplicadas a solução corrente, afetando assim a exploração e a busca de forma direta. No artigo original proposto por Dubois-Lacoste, Pagnozzi e Stützle (2017), foram testados valores de 0 a 10 para K, onde o melhor resultado obtido foi $k = 2$.

Da mesma forma como realizado com o parâmetro k, também foi feito com o parâmetro T_p, onde foram testados valores entre 0.0 e 1.5, com intervalo de 0.1 entre cada valor. O melhor parâmetro obtido foi $T_p = 0.7$.

Tabela 14 – Parâmetros da heurística IG_{BLSP}

Parâmetro	**Valor**
K	2
T_p	0.7

4.2.8 Exemplo numérico

O exemplo aqui apresentado seguirá o problema de escalonamento contido na Tabela 8, que está presente na seção 4.1.2. O critério de parada para este exemplo é 1 (uma) execução do algoritmo IG_{BLSP}.

4.2.8.1 Solução inicial

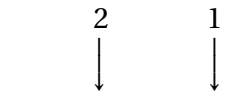
A solução inicial para este exemplo está presente de forma mais detalhada na tabela 13b e será representada da seguinte maneira:

$$\pi = \boxed{J_1 \mid J_3 \mid J_5 \mid J_4 \mid J_2} \qquad C_{max} = 495$$

Faça: $\pi^* \leftarrow \pi$

4.2.8.2 Fase de destruição

Dada a solução inicial, temos então a fase de destruição.



$\pi =$ | J_1 | J_3 | J_5 | J_4 | J_2 | *K=2* trabalhos escolhidos aleatoriamente

$\pi_R =$ | J_1 | J_5 | J_2 | Sequência parcial para reconstrução ($C_{max} = 435$)

$\pi_D =$ | J_4 | J_3 | Removidos da permutação para ser reinseridos

4.2.8.3 Busca local na solução parcial

A BLSP será feira na sequência π_R.

Seja: $j = J_1$ (escolhido aleatoriamente na sequência π_R)

| J_1 | J_5 | J_2 | $C_{max} = 425$

| J_5 | J_1 | J_2 | $\boldsymbol{C_{max} = 424}$

| J_5 | J_2 | J_1 | $C_{max} = 435$

$\pi_{aux} =$ | J_5 | J_1 | J_2 | (Melhor permutação encontrada)

Como $\boldsymbol{C_{max}(\pi_{aux}) < C_{max}(\pi_R)}$ então $\pi_R \leftarrow \pi_{aux}$ e escolhe outro *j*.

Seja: $j = J_2$ (escolhido aleatoriamente na sequência π_R)

| J_2 | J_5 | J_1 | $C_{max} = 449$

| J_5 | J_2 | J_1 | $C_{max} = 435$

| J_5 | J_1 | J_2 | $\boldsymbol{C_{max} = 424}$

$\pi_{aux} =$ | J_5 | J_1 | J_2 | (Melhor permutação encontrada)

Como $\boldsymbol{C_{max}(\pi_{aux}) = C_{max}(\pi_R)}$ então retorna π_R.

4.2.8.4 Fase de Construção

Deverá ser feita a inserção em π_R dos trabalhos contidos em π_D.

Seja $\pi_D(1) = J_4$, então a permutação em π_R será:

J_4	J_5	J_1	J_2	$C_{max} = 517$
J_5	J_4	J_1	J_2	$\boldsymbol{C_{max} = 492}$
J_5	J_1	J_4	J_2	$C_{max} = 492$
J_5	J_1	J_2	J_4	$C_{max} = 492$

$\pi_R =$ | J_5 | J_4 | J_1 | J_2 | (Melhor permutação encontrada)

Após a inserção do trabalho J_4 em π_R e a sua respectiva remoção de π_D temos que:

$\pi_D =$ | J_3 | (Trabalhos para serem inseridos)

Logo temos que $\pi_D(1) = J_3$ então a permutação em π_R será:

J_3	J_5	J_4	J_1	J_2	$\boldsymbol{C_{max} = 507}$
J_5	J_3	J_4	J_1	J_2	$C_{max} = 523$
J_5	J_4	J_3	J_1	J_2	$C_{max} = 523$
J_5	J_4	J_1	J_3	J_2	$C_{max} = 538$
J_5	J_4	J_1	J_2	J_3	$C_{max} = 523$

$\pi_R =$ | J_3 | J_5 | J_4 | J_1 | J_2 | (Melhor permutação encontrada)

Após a inserção do trabalho J_3 em π_R e a sua respectiva remoção de π_D temos que $\pi_D = \emptyset$, logo esta etapa está concluída e então $\pi' \leftarrow \pi_R$.

4.2.8.5 Busca local na solução completa

$$\pi' = \boxed{J_3 \mid J_5 \mid J_4 \mid J_1 \mid J_2} \qquad \boldsymbol{C_{max}} = \mathbf{507}$$

Seja: $j = J_2$ (escolhido aleatoriamente na sequência π')

$$\pi' - j = \boxed{J_3 \mid J_5 \mid J_4 \mid J_1}$$

Ao permutar j em $\pi' - j$ temos que:

$$\pi_{aux} = \boxed{J_2 \mid J_3 \mid J_5 \mid J_4 \mid J_1} \qquad C_{max} = 532$$

Como $\boldsymbol{C_{max}}(\boldsymbol{\pi_{aux}}) \geq \boldsymbol{C_{max}}(\boldsymbol{\pi}')$ então continue a permutação de j:

$$\pi_{aux} = \boxed{J_3 \mid J_2 \mid J_5 \mid J_4 \mid J_1} \qquad C_{max} = 532$$

Como $\boldsymbol{C_{max}}(\boldsymbol{\pi_{aux}}) \geq \boldsymbol{C_{max}}(\boldsymbol{\pi}')$ então continue a permutação de j:

$$\pi_{aux} = \boxed{J_3 \mid J_5 \mid J_2 \mid J_4 \mid J_1} \qquad C_{max} = 518$$

Como $\boldsymbol{C_{max}}(\boldsymbol{\pi_{aux}}) \geq \boldsymbol{C_{max}}(\boldsymbol{\pi}')$ então continue a permutação de j:

$$\pi_{aux} = \boxed{J_3 \mid J_5 \mid J_4 \mid J_2 \mid J_1} \qquad C_{max} = 507$$

Como $\boldsymbol{C_{max}}(\boldsymbol{\pi_{aux}}) \geq \boldsymbol{C_{max}}(\boldsymbol{\pi}')$ então continue a permutação de j:

$$\pi_{aux} = \boxed{J_3 \mid J_5 \mid J_4 \mid J_1 \mid J_2} \qquad C_{max} = 507$$

Como $\boldsymbol{C_{max}}(\boldsymbol{\pi_{aux}}) \geq \boldsymbol{C_{max}}(\boldsymbol{\pi}')$ e não existe mais permutação de j, então retorna π'.

4.2.8.6 Critério de aceitação

Como $\boldsymbol{C_{max}}(\boldsymbol{\pi}') = \boldsymbol{C_{max}}(\boldsymbol{\pi})$ então $\pi \leftarrow \pi'$.

4.2.8.7 Atualização do melhor global

Como $\boldsymbol{C_{max}(\pi')} \geq \boldsymbol{C_{max}(\pi^*)}$ então não tem alteração no melhor global.

Por fim temos que:

$$\pi = \boxed{J_3 \mid J_5 \mid J_4 \mid J_1 \mid J_2}$$

$$\pi^* = \boxed{J_1 \mid J_3 \mid J_5 \mid J_4 \mid J_2}$$

4.2.8.8 Critério de parada

Para este exemplo o critério de parada escolhido foi de **uma execução** do algoritmo, logo este critério já foi atingido e o resultado final do algoritmo está contido em $\boldsymbol{\pi^*}$.

Observação: Se o critério de parada ainda não tivesse sido atingido, o algoritmo voltaria no passo de Destruição localizado na seção 4.2.8.2 com o parâmetro $\boldsymbol{\pi}$.

Portanto a resposta final do algoritmo é a permutação:

$$\boxed{\boldsymbol{J_1} \mid \boldsymbol{J_3} \mid \boldsymbol{J_5} \mid \boldsymbol{J_4} \mid \boldsymbol{J_2}} \qquad \boldsymbol{C_{max} = 495}$$

4.3 Busca do Macaco Híbrido

O algoritmo Busca do Macaco Híbrido (BMH) foi proposto por Marichelvam, Tosun e Geetha (2017) com o objetivo de resolver o PFSP e é baseado no método *Monkey Algorithm* (*MA*) desenvolvido por Zhao e Tang (2008) para solucionar problemas globais de otimização numérica.

O método deriva da simulação do processo de escalada da montanha por macacos. Suponha que existam muitas montanhas em um determinado campo (ou seja, o espaço de busca), para encontrar o topo mais alto (isto é, encontrar o melhor valor para a função objetivo, menor C_{max}), os macacos escalarão a montanha de suas respectivas posições (esta ação é chamada de processo de escalada).

Para cada macaco, quando chega ao topo da montanha, é natural observar e descobrir se existem outras montanhas à sua volta acima da posição em que está atualmente. Se sim, ele vai saltar em algum lugar da montanha observada por ele a partir da posição atual (essa ação é chamada de processo de observar-saltar) e depois repetirá o processo de escalada até atingir o topo da montanha.

Após as repetições do processo de escalada e do processo de observar-saltar, cada macaco encontrará um topo da montanha local. Para encontrar um topo de montanha mais alto, é natural que cada macaco passe por uma nova etapa de busca (essa ação é chamada de processo de salto mortal).

Depois de várias repetições do processo de escalada, do processo de observar-saltar e do processo de salto mortal, o topo mais alto encontrado pelos macacos será relatado como o melhor valor encontrado.

De acordo com o Algoritmo 4.3 que contém o pseudocódigo do BMH, a entrada contem *M* soluções, onde *M-1* são soluções geradas aleatoriamente com valores reais no intervalo [0,1] e a outra solução restante é gerada pelo algoritmo NEH desenvolvido por Nawaz, Enscore e Ham (1983) e está descrito na seção 4.1. A saída é a melhor solução encontrada durante a execução do BMH.

Algoritmo 4.3: PSEUDOCÓDIGO BUSCA DO MACACO HÍBRIDO

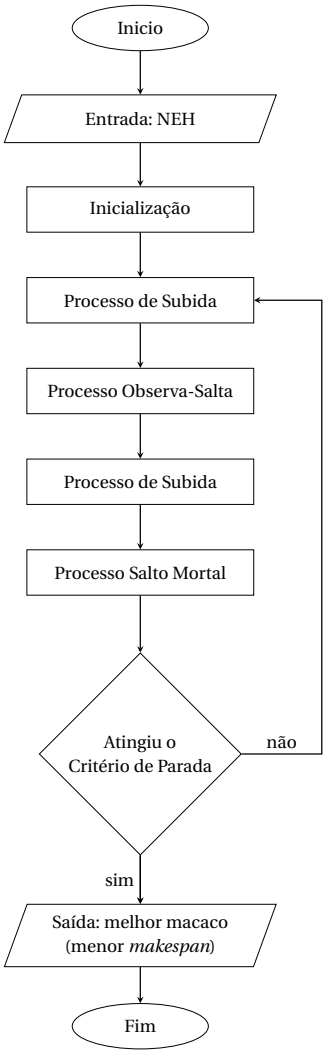
Entrada: Solução inicial π `/* M-1 soluções aleatórias e 1 gerada pela heurística NEH */`
Saída: A melhor solução encontrada π^*

```
1  início
2      π* ← Melhor_Inicial(π);
3      enquanto diferente do critério de parada faça
4          π_b ← Processo_Subida(π);
5          π_j ← Processo_Observa_Salta(π_b);
6          π_x ← Processo_Subida(π_j);
7          π ← Processo_Salto_Mortal(π_x);
8          π* ← Melhor_Global(π*, π_b, π_j, π_x, π);
9      fim
10 fim
11 retorna π*
```

Observando o algoritmo em si, no laço principal enquanto o critério de parada não for atingido, podemos ver que na linha 4 temos o primeiro processo de subida, a partir da solução inicial. Na linha 5 será feita a etapa de observar-saltar, tendo como entrada a saída da etapa anterior. O segundo processo de subida, linha 6, montará uma solução a partir do observar-saltar feito anteriormente. Após isso, linha 7, será realizado o processo do salto mortal na solução encontrada na fase anterior. Por fim, na linha 8, a melhor solução será calculada com base nas saídas de todas as etapas anteriores.

Na Figura 5 foi apresentado um fluxograma que descreve visualmente os passos do BMH. Os detalhes e os parâmetros da implementação do algoritmo, bem como um exemplo numérico estão apresentados nas subseções a seguir.

Figura 5 – Fluxograma da meta-heurística Busca do Macaco Híbrida



4.3.1 Representação da solução

Primeiramente, um inteiro M será definido como sendo o tamanho da população de macacos. Então, para cada $i \in \{1,2,...,M\}$, será definido um macaco $X_i = [x_{i1}, x_{i2}, ..., x_{in}]$ que corresponde a posições de valores contínuos, onde n representa o número de dimensões, sendo que cada dimensão equivale a um trabalho do PFSP. Este macaco expressa então uma solução do problema de otimização PFSP.

4.3.2 População Inicial

Para iniciar o algoritmo primeiro se cria uma população inicial. Para isto a inicialização da posição de cada macaco é necessária. Cada trabalho j do PFSP é como se fosse uma dimensão da posição e tem um valor associado escolhido aleatoriamente dentro do intervalo [0,1]. Como visto na seção anterior, tem-se um total de M macacos, e portanto M soluções possíveis. Destas M, uma será gerada pela heurística NEH e as outras M-*1* serão

geradas aleatoriamente. Após criadas as M soluções iniciais, a população inicial ficará da seguinte maneira: $P_{init} = [A_1, A_2, ..., A_{M-1,N}]$.

4.3.2.1 NEH

Como a heurística NEH já retorna uma sequência de j trabalhos $Y = [y_1, y_2, ..., y_j]$, basta criar uma sequência $Z = [z_1, z_2, ..., z_j]$ com j valores reais aleatórios entre o intervalo $[0,1]$ ordenados de forma crescente. Estes valores de Z serão então associados a sequência de trabalhos do NEH. Assim a solução $N = [y_1 : z_1, y_2 : z_2, ..., y_j : z_j]$ é criada e adicionada ao conjunto de soluções iniciais representando a heurística NEH,.

4.3.2.2 Aleatórias

Para cada macaco $i \in \{1,2,...,M-1\}$ cria-se uma sequência $Y_i = [y_{i1}, y_{i2}, ..., y_{ij}]$ de trabalhos e uma sequência $Z_i = [z_{i1}, z_{i2}, ..., z_{ij}]$ com valores reais aleatórios entre o intervalo $[0,1]$. Então a sequência de trabalhos Y_i é associada ao vetor Z_i. O resultado desta associação é a sequência $A_i = [y_{i1} : z_{i1}, y_{i2} : z_{i2}, ..., y_{ij} : z_{ij}]$.

Com o objetivo de converter as posições dos macacos em permutações dos trabalhos de maneira discreta, a regra Menor Valor de Posição [1] (MVP) é aplicada em cada sequência A_i. O MVP vai ordenar de forma crescente a sequência A_i pelos valores associados a cada trabalho. No total são criadas *M-1* soluções aleatórias que serão ordenadas pela regra MVP e elas serão adicionadas ao conjunto de soluções iniciais.

4.3.3 Processo de subida

Este é um procedimento que muda passo-a-passo as posições dos macacos partindo da solução inicial ate chegar em novas soluções, que podem melhorar o valor de C_{max}. Para entender este processo, algumas definições serão explicadas a baixo:

- *Macaco:* X_i^t denota o i-ésimo macaco da população dentro da iteração t. É representado por um número n de dimensões, sendo $X_i^t = \left[x_{i1}^t, x_{i2}^t, ..., x_{in}^t\right]$, onde x_{ij}^t é o valor de posição do i-ésimo macaco com respeito a j-ésima dimensão com $j \in \{1,2,...,n\}$.
- *População:* X^t é o conjunto de M macacos na iteração t, sendo $X^t = \left[X_1^t, X_2^t, ..., X_M^t\right]$.
- *Permutação:* Uma nova variável π_i^t, que é uma permutação implícita do macaco X_i^t, é aqui introduzida. Ela pode ser descrita como $\pi_i^t = \left[\pi_{i1}^t, \pi_{i2}^t, ..., \pi_{in}^t\right]$, onde π_{ij}^t é a atribuição do trabalho j ao macaco i sobre a permutação na iteração t, com $j \in \{1,2,...,n\}$.

[1] Tradução direta do inglês de *Smallest Position Value*

- *Velocidade do macaco:* V_i^t é a velocidade do macaco i na na iteração t. Ela pode ser definida como $V_i^t = \left[v_{i1}^t, v_{i2}^t, ..., v_{in}^t\right]$, onde v_{ij}^t é a velocidade do macaco i na na iteração t com respeito a j-ésima dimensão.

- *Peso de inércia e coeficientes de aceleração:* O parâmetro w^t controla o impacto das velocidades anteriores na velocidade atual. Ele tem um impacto na escolha de um em detrimento do outro entre as capacidades globais e locais de exploração do macaco. No início da busca, um valor de peso de inércia alto é usado para melhorar a exploração global, enquanto é reduzido no decorrer das buscas para que haja uma melhor exploração local mais tarde. Os coeficientes de aceleração *c1* e *c2* são parâmetros constantes que controlam o tamanho do passo máximo que o macaco pode dar.

- *Melhor local:* P_i^t representa a melhor posição do macaco i com o melhor valor de *makespan* até a iteração t, então a melhor posição associada ao melhor *makespan* do macaco até o momento é chamado de melhor pessoal. Para cada macaco na busca, P_i^t pode ser determinado e atualizado em cada iteração t. Para simplificar a terminologia o *makespan* $\left(C_{max}()\right)$ é tratado aqui como $f()$. No problema *flowshop*, onde queremos minimizar $f\left(\pi_i^t\right)$ e π_i^t corresponde a permutação do macaco X_i^t, o melhor local P_i^t do i-ésimo macaco será atualizado com o valor de π_i^t se $f\left(\pi_i^t\right) \leq f\left(P_i^{t-1}\right)$, senão P_i^t será atualizado com o valor de P_i^{t-1}, onde P_i^{t-1} corresponde ao melhor local encontrado ate a iteração anterior. Para simplificar ainda mais, a função de avaliação do melhor local será denotada como $f_i^{ml} = f\left(\pi_i^t\right)$. Para cada macaco, o melhor local é definido como $P_i^t = \left[p_{i1}^t, p_{i2}^t, ..., p_{ij}^t\right]$, onde p_{in}^t é o valor de posição do i-ésimo melhor local com respeito a j-ésima dimensão com $j \in \{1, 2, ..., n\}$.

- *Melhor global:* G^t indica a posição do melhor macaco alcançado até a iteração t em toda a busca. Se $min\left(f_i^{ml}\right) \leq G^{t-1}$ com $i \in \{1, 2, ..., M\}$ então G^t será atualizado com $min\left(f_i^{ml}\right)$, senão, G^t irá ser atualizado com a permutação presente em G^{t-1}. Para simplificar, a função de avaliação do melhor global será denotada como $f^{mg} = f\left(\pi^t\right)$, onde π^t corresponde a permutação de G^t. O melhor global é definido como $G^t = \left[g_1^t, g_2^t, ..., g_n^t\right]$, onde g_j^t é o valor de posição do melhor global com respeito a j-ésima dimensão com $j \in \{1, 2, ..., n\}$.

- *Critério de parada da subida:* define qual a condição que irá parar o processo de subida. Neste caso o critério de parada será o número de subidas do macaco, que será chamado de N_{sub} e está definido na seção 4.3.7.

- *Critério de parada do algoritmo:* define em qual condição a execução do BMH será interrompida. Aqui o critério de parada será o número de iterações do algoritmo, que será chamado de N_{iter} e está definido na seção 4.3.7.

4.3.3.1 Inicialização

A população inicial dos macacos para este processo está definida na seção 4.3.2.

As velocidades iniciais de cada posição dos macacos contidos na solução inicial é estabelecida de maneira aleatória. A seguinte formula é utilizada para gerar as velocidades iniciais:

$$v_{ij}^{0} = v_{min} + \left(v_{max} - v_{min}\right) \cdot a \tag{4.3}$$

onde $v_{min} = -4.0$, $v_{max} = 4.0$ e a é um número aleatório uniforme entre 0 e 1.

Os valores das velocidades estão continuamente restritos ao intervalo:

$$v_{ij}^{t} = \left[v_{min}, v_{max}\right] = \left[-4.0, 4.0\right]$$

onde $v_{min} = -v_{max}$. Isto quer dizer que os valores das velocidades nunca podem sair deste intervalo, nem mesmo em outras iterações do processo de subida.

4.3.3.2 Procedimento Computacional

O procedimento computacional completo pode ser resumido nos seguintes passos:

4.3.3.2.1 Inicialização

- Faça $t \leftarrow 0$.
- Se $iter = 1$ então faça $X^0 \leftarrow$ *população inicial* (definida na seção 4.3.2), senão faça $X^0 \leftarrow$ *população final* da iteração $iter - 1$ (definida na seção 4.3.5).
- Crie aleatoriamente as velocidades iniciais $\{V_i^0\}$ para cada macaco, onde $i \in \{1, 2, ..., M\}$.
- Faça $P_i^0 \leftarrow X_i^0$, onde $P_i^0 = [p_{i1}^0 = x_{i1}^0, p_{i2}^0 = x_{i2}^0, ..., p_{ij}^0 = x_{ij}^0]$, juntamente com o valor de f_i^{ml} para $i \in \{1, 2, ..., M\}$ e $j \in \{1, 2, ..., n\}$.
- Encontre o melhor *makespan* de forma que $f_k \leftarrow \min(f_i^{ml})$ para $i \in \{1, 2, ..., M\}$, com as correspondentes posições X_k^0. Para o melhor global faça $G^0 \leftarrow X_k^0$, onde $G^0 = [g_1 = x_{k,1}, g_2 = x_{k,2}, ..., g_n = x_{k,n}]$, juntamente com $f^{mg} \leftarrow f_k$.

4.3.3.2.2 Atualização da iteração

$$t = t + 1 \tag{4.4}$$

4.3.3.2.3 Atualização do peso de inercia

$$w^t = w^{t-1} \cdot \beta \tag{4.5}$$

onde β é o fator de decremento e está definido nos parâmetros na seção 4.3.7.

4.3.3.2.4 Atualização da velocidade

O método para a atualização foi baseado na atualização de velocidades do PSO que pode ser encontrado em Tasgetiren et al. (2007). Isto é importante pois indica com qual velocidade os macacos tentarão atingir a posição mais alta da montanha.

$$v_{ij}^t = \left(w^{t-1} \cdot v_{ij}^{t-1}\right) + \left[c_1 \cdot a_1 \cdot \left(p_{ij}^{t-1} - x_{ij}^{t-1}\right)\right] + \left[c_2 \cdot a_2 \cdot \left(g_j^{t-1} - x_{ij}^{t-1}\right)\right] \tag{4.6}$$

onde c_1 e c_2 são os coeficientes de aceleração (definidos na tabela 15) e a_1 e a_2 são números aleatórios uniformes entre o intervalo [0, 1].

4.3.3.2.5 Atualização da posição

A obtenção da posição do macaco na iteração atual (t) do processo de subida é feita mediante a soma da posição na iteração anterior (t-1) com a velocidade na iteração atual. A atualização é obtida da seguinte forma:

$$x_{ij}^t = x_{ij}^{t-1} + v_{ij}^t \tag{4.7}$$

4.3.3.2.6 Ordenando a posição

Após a atualização das posições, estas podem estar desordenadas, então se deve aplicar a regra do MVP em X_i^t com a finalidade de encontrar uma nova permutação X_i^t ordenada para cada macaco $i \in \{1, 2, ..., M\}$.

4.3.3.2.7 Atualização do melhor local

Cada macaco será avaliado a partir de sua permutação para ver se o melhor local até a iteração anterior vai será melhorado, isto é, se $C_{max}(X_i^t) < f_i^{ml}$, para $i \in \{1, 2, ..., M\}$, então o melhor local será atualizado com $P_i^t \leftarrow X_i^t$ e $f_i^{ml} \leftarrow C_{max}(X_i^t)$. Senão os valores de P_i^t e f_i^{ml} continuarão os mesmos da iteração anterior.

4.3.3.2.8 Atualização do melhor global

1. Encontre o menor valor de f dentre os melhores locais, isto é, $f_k^t \leftarrow \min(f_i^{ml})$, onde $i \in \{1,2,...,M\}$ e k é o índice i do menor f^{ml}.
2. Se $f_k^t < f^{mg}$, então o melhor global será atualizado com $G^t \leftarrow P_k^t$ e $f^{mg} \leftarrow f_k^t$. Senão os valores de G^te f^{mg} continuarão os mesmos da iteração anterior.

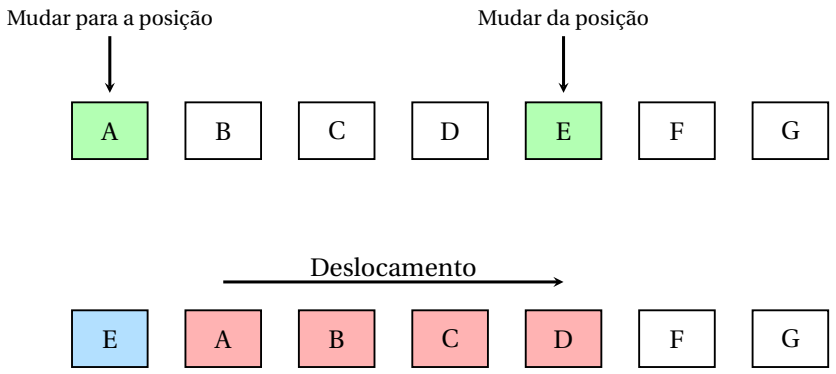
4.3.3.2.9 Mutação

Para prevenir que a solução encontrada ate aqui no processo de subida fique preso em um ótimo local, um operador de mutação é aplicado neste ponto do algoritmo. O operador aqui utilizado é o mudança por deslocamento [2] com probabilidade de acontecer igual a $Prob_m$, onde $Prob_m$ é um parâmetro e está definido na seção 4.3.7.

Caso a mudança por deslocamento ocorra (de acordo com a probabilidade), ela será efetuada no melhor macaco encontrado ate aqui, ou seja, no melhor global G^t. Um exemplo de mutação mudança por deslocamento pode ser visto na Figura 6 e ela acontece da seguinte maneira:

1. Escolha dois números *(pos1 e pos2)* aleatórios e não repetidos no intervalo $(1,n)$.
2. Faça $Aux \leftarrow G^t$.
3. Remova de Aux o trabalho que está contido na posição *pos1*.
4. Pegue o trabalho removido no passo anterior e o reinsira em Aux na posição *pos2*.
5. Se $C_{max}(Aux) < f_{mg})$ então faça $G^t \leftarrow Aux$ e atualize o valor do melhor local na posição k com $P_k^t \leftarrow Aux$, onde k é o índice encontrado na seção 4.3.3.2.8.
6. Senão, G^t e P_k^t não sofrerão alterações.

Figura 6 – Mutação mudança por deslocamento utilizada no algoritmo BMH



[2] Tradução direta do inglês de *Shift Change*

4.3.3.2.10 Critério de parada

- Se $t < N_{sub}$ (o parâmetro N_{sub} está descrito na seção 4.3.7) então volte ao passo descrito na seção 4.3.3.2.2.
- Senão, faça $X_i \leftarrow P_i$ para cada macaco $i \in \{1,2,...,M\}$, pare o processo de subida e vá para a próxima etapa do algoritmo.

4.3.4 Processo observa-salta

Depois do processo de subida, os macacos chegarão em um ponto máximo de suas respectivas montanhas. Então cada um deles irá observar em volta e determinar se existe algum outro ponto que seja mais alto que a posição atual em que se encontra. Se sim, o macaco vai saltar da posição atual para esta nova posição. Para isso os macacos terão uma visão, que indica qual é a distancia máxima que os macacos poderão observar. Cada macaco poderá observar um número máximo de vezes definido pelo parâmetro W_{obs}.

Uma observação importante é que este parâmetro não está descrito no artigo original e ao entrar em contato com o autor original ele disse que não poderia revelar o valor deste parâmetro por motivos comerciais, por isso ele deixou aberta uma margem para interpretação. Então o valor de W_{obs} definido nesta monografia não representa o valor do artigo original e sim o do autor desta monografia. Após a realização de testes, o melhor valor para este parâmetro foi escolhido e se encontra definido na seção 4.3.7. Já a visão que cada macaco tem foi descrita no artigo original. Ela é definida como um parâmetro positivo b, e o seu valor também está representado na seção 4.3.7.

Para cada macaco i, sendo que $i \in \{1,2,...,M\}$, faça os seguintes passos:

1. Faça $z \leftarrow 0$
2. Crie aleatoriamente uma lista de números reais y_j onde cada número aleatório deve estar compreendido no intervalo $(x_{ij} - b, x_{ij} + b)$ para $j \in \{1,2,...,n\}$.
   Faça $y \leftarrow (y_1, y_2, ..., y_j)$.
3. Aplique a regra do MVP em y e obtenha um novo y ordenado.
4. Se $C_{max}(y) \leq C_{max}(x_i)$ faça $X_i \leftarrow y$. Senão, caso $z < W_{obs}$ então faça $z \leftarrow z + 1$, repita novamente o processo a partir do passo 2. Caso contrário, ou seja, se $z \geq W_{obs}$ o valor de X_i continuará o mesmo.

Quando os passos descritos acima forem realizados para todos os macacos, então repita o processo de subida, descrito na seção 4.3.3, tendo os valores de $\boldsymbol{x}$, definidos no passo 4, como população inicial. Ao final deste processo de subida vá para o processo do salto mortal descrito na seção 4.3.5.

4.3.5 Processo do Salto Mortal

O principal objetivo do processo do salto mortal é deixar os macacos explorar novos espaços de busca. O baricentro das posições correntes de todos os macacos será chamado de pivô. Então os macacos executarão o salto mortal direcionados pelo pivô. Mais especificamente, cada macaco i vai fazer o salto mortal para um novo ponto na montanha a partir da sua posição atual X_i da seguinte maneira, para cada macaco i, sendo que $i \in \{1,2,...,M\}$:

1. Escolha um número aleatório α presente no intervalo $[c,d]$ (chamado de intervalo do salto mortal), onde c e d estão definidos na seção 4.3.7.
2. Faça:
$$y_j = x_{ij} + \alpha \cdot (p_j - x_{ij}) \tag{4.8}$$
onde
$$p_j = \frac{1}{M} \cdot \sum_{i=1}^{M} x_{ij} \tag{4.9}$$
com $j \in \{1,2,...,n\}$. O ponto $\boldsymbol{p} = (p_1, p_2, ..., p_n)$ é chamado de pivô do salto mortal.
3. Faça $y \leftarrow (y_1, y_2, ..., y_j)$.
4. Aplique a regra do MVP em y e obtenha um novo y ordenado.
5. Faça $X_i \leftarrow y$.

4.3.6 Critério de parada

Os processos descritos anteriormente se repetirão de maneira cíclica ate que um critério de parada seja atingido. Os valores de $\boldsymbol{X}$, definidos no passo 5 da seção 4.3.5, serão tidos como os novos macacos, ou seja, a população inicial nas novas iterações do BMH. A execução do algoritmo vai parar quando forem atingidos N_{iter} ciclos de repetição, onde N_{iter} é um parâmetro e está definido na seção 4.3.7, então será retornado no fim da execução o macaco que atingiu a posição mais alta na montanha em algum ciclo, ou seja, o macaco com o menor *makespan* encontrado entre todos os ciclos.

4.3.7 Parâmetros

Na tabela 15 estão descritos o nome, a sigla e o valor de cada um dos parâmetros utilizados no BMH.

Tabela 15 – Parâmetros da meta-heurística BMH

Nome	**Sigla**	**Valor**
Tamanho da população	M	20
Coeficientes de aceleração	$c_1 = c_2$	2
Peso de inercia inicial	w^0	0.9
Fator de decremento	β	0.975
Intervalo do peso de inercia	w^t	$[0.4,\ 0.9]$
Número de subidas	N_{sub}	2000
Probabilidade de mutação	$Prob_m$	0.01 (1%)
Número de observações	W_{obs}	1000
Visão do macaco	b	0.5
Intervalo do salto mortal	$[c, d]$	$[-1,\ 1]$
Ciclos de repetição (iterações)	N_{iter}	200

4.3.8 Exemplo numérico

O exemplo aqui apresentado seguirá o problema de escalonamento contido na tabela 8. Alguns parâmetros foram modificados para este exemplo para que este ficasse curto e de fácil compreensão, os parâmetros modificados estão contidos na tabela 16, já os demais não sofreram alterações e continuam os mesmos da tabela 15. Com a finalidade de simplificar os cálculos foram utilizadas duas casas decimais.

Tabela 16 – Parâmetros do exemplo para o algoritmo BMH

Nome	**Sigla**	**Valor**
Tamanho da população	M	3
Ciclos de repetição (iterações)	N_{iter}	2
Número de subidas	N_{sub}	3
Número de observações	W_{obs}	4

1° Ciclo de repetição do BMH

No inicio tem-se que $iter \leftarrow 1$, onde *iter* é a variável correspondente ao atual ciclo de repetição (iteração) em que se encontra o algoritmo.

4.3.8.1 População inicial

A população inicial, representada por X^0, é composta por duas soluções aleatórias $(A_1$ e $A_2)$ e uma solução gerada pela heurística inicial NEH. Sendo assim temos que: $P_{init} = (A_1, A_2, N)$

População inicial aleatória

$A_1 =$	$J_1 : 0.38$	$J_4 : 0.93$	$J_5 : 0.48$	$J_3 : 0.56$	$J_2 : 0.81$
$A_2 =$	$J_4 : 0.44$	$J_1 : 0.82$	$J_5 : 0.89$	$J_2 : 0.56$	$J_3 : 0.66$

Aplicando a regra MVP em $A_1 e A_2$ temos que:

$A_1 =$	$J_1 : 0.38$	$J_5 : 0.48$	$J_3 : 0.56$	$J_2 : 0.81$	$J_4 : 0.93$
$A_2 =$	$J_4 : 0.44$	$J_2 : 0.56$	$J_3 : 0.66$	$J_1 : 0.82$	$J_5 : 0.89$

População inicial NEH

Foi criado aleatoriamente um vetor $v = [0.28, 0.09, 0.81, 0.23, 0.08]$. Este vetor foi ordenado utilizando a regra MVP, obtendo-se um novo vetor $v = [0.08, 0.09, 0.23, 0.28, 0.81]$.

A solução encontrada pela heurística NEH pode ser encontrada de forma detalhada na tabela 13b, tem como resultado a sequência $[J_1, J_3, J_5, J_4, J_2]$. Depois foi feita a associação de v com a solução encontrada por NEH da seguinte maneira:

N =	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$

População inicial P_{init}

Por fim têm-se que P_{init} é igual a:

$A_1 =$	$J_1 : 0.38$	$J_5 : 0.48$	$J_3 : 0.56$	$J_2 : 0.81$	$J_4 : 0.93$
$A_2 =$	$J_4 : 0.44$	$J_2 : 0.56$	$J_3 : 0.66$	$J_1 : 0.82$	$J_5 : 0.89$
N =	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$

4.3.8.2 Processo de subida

Aqui será mostrado o processo de subida.

1° Subida

Inicialização

$t \leftarrow 0$, onde t é a variável correspondente a atual subida em que se encontra o processo de subida.

Velocidade inicial

$V_1^0 =$	0.72	0.88	−1.3	−3.97	1.39
$V_2^0 =$	−2.48	−1.53	−2.81	0.33	−2.73
$V_3^0 =$	0.88	2.98	0.85	−1.64	−3.99

População inicial

Como $iter = 1$ então $X^0 \leftarrow P_{init}$.

$X_1^0 =$	$J_1 : 0.38$	$J_5 : 0.48$	$J_3 : 0.56$	$J_2 : 0.81$	$J_4 : 0.93$
$X_2^0 =$	$J_4 : 0.44$	$J_2 : 0.56$	$J_3 : 0.66$	$J_1 : 0.82$	$J_5 : 0.89$
$X_3^0 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$

Melhor local

$P^0 \leftarrow X^0$.

$f_1^{ml} \leftarrow C_{max}(X_1^0) = 542$.

$f_2^{ml} \leftarrow C_{max}(X_2^0) = 522$.

$f_3^{ml} \leftarrow C_{max}(X_3^0) = 495$.

$P_1^0 =$	$J_1 : 0.38$	$J_5 : 0.48$	$J_3 : 0.56$	$J_2 : 0.81$	$J_4 : 0.93$	$f_1^{ml} = 542$
$P_2^0 =$	$J_4 : 0.44$	$J_2 : 0.56$	$J_3 : 0.66$	$J_1 : 0.82$	$J_5 : 0.89$	$f_2^{ml} = 525$
$P_3^0 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$	$f_3^{ml} = 495$

Melhor global

$f_k \leftarrow \min(f_1^{ml}, f_2^{ml}, f_3^{ml}) \quad = \quad \min(542, 525, 495) \quad = \quad 495$

Como o macaco com o menor C_{max} é o 3, então $G^0 \leftarrow P_3^0$ e $f^{mg} \leftarrow f_k$.

$G^0 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$	$f^{mg} = 495$

Atualização da iteração

$t \leftarrow t + 1 \quad = \quad 0 + 1 \quad = \quad 1$

Atualização do peso de inercia

$w^1 \leftarrow w^0 \cdot \beta \quad = \quad 0.9 \cdot 0.975 \quad = \quad 0.87$

Atualização das velocidades

A demonstração da atualização da velocidade será feita somente para v_{11}^1 pois os cálculos das demais velocidades são feitos de forma semelhante.

$a_1 \leftarrow \text{aleatório}(0\ ,\ 1) = 0.08$

$a_2 \leftarrow \text{aleatório}(0\ ,\ 1) = 0.5$

Seguindo a equação 4.7 têm-se:

$v_{11}^1 = (0.9 \cdot 0.72) + [2 \cdot 0.08 \cdot (0.38 - 0.38)] + [2 \cdot 0.5 \cdot (0.08 - 0.38)]$

$v_{11}^1 = 0.64 + 0 - 0.3$

$v_{11}^1 = 0.34$

Análogo ao exemplo acima, atualizando as demais velocidades obtém-se:

$V_1^1 =$	0.34	−1.97	−0.26	−2.09	−2.32
$V_2^1 =$	−2.36	−2.15	−0.72	−1.8	−0.24
$V_3^1 =$	−0.59	1.26	−0.34	−3.13	−1.55

Atualização das posições

A demonstração da atualização da posição será feita somente para X_{11}^1 pois os cálculos das demais posições são feitos de forma semelhante.

Seguindo a equação 4.6 têm-se:

$x_{11}^1 = 0.38 + 0.34$

$x_{11}^1 = 0.72$

Análogo ao exemplo acima, atualizando as demais posições obtém-se:

$X_1^1 =$	$J_1 : 0.72$	$J_5 : -1.49$	$J_3 : 0.3$	$J_2 : -1.27$	$J_4 : -1.38$
$X_2^1 =$	$J_4 : -1.92$	$J_2 : -1.59$	$J_3 : -0.05$	$J_1 : -0.98$	$J_5 : 0.65$
$X_3^1 =$	$J_1 : -0.51$	$J_3 : 1.35$	$J_5 : -0.11$	$J_4 : -2.84$	$: j_2 : -0.74$

Ordenando as posições

Ao aplicar o MVP em X_1^1 têm-se:

$X_1^1 =$	$J_5 : -1.49$	$J_4 : -1.38$	$J_2 : -1.27$	$J_3 : 0.3$	$J_1 : 0.72$	$C_{max} = 523$
$X_2^1 =$	$J_4 : -1.92$	$J_2 : -1.59$	$J_1 : -0.98$	$J_3 : -0.05$	$J_5 : 0.65$	$C_{max} = 523$
$X_3^1 =$	$J_4 : -2.84$	$J_2 : -0.74$	$J_1 : -0.51$	$J_5 : -0.11$	$J_3 : 1.35$	$C_{max} = 541$

Atualização do melhor local

Tendo sido os valores de P_i^0 e f_i^{ml} definidos na iteração 0 e X_i^1 e $C_{max}(X_1^1)$ na seção anterior, onde $i \in \{1, 2, 3\}$, tem-se que:

Como $C_{max}(X_1^1) < f_1^{ml}$, então $P_1^1 \leftarrow X_1^1$ e $f_1^{ml} \leftarrow C_{max}(X_1^1)$.

Como $C_{max}(X_2^1) < f_2^{ml}$, então $P_2^1 \leftarrow X_2^1$ e $f_2^{ml} \leftarrow C_{max}(X_2^1)$.

Como $C_{max}(X_3^1) > f_3^{ml}$, então $P_3^1 \leftarrow P_3^0$ e $f_3^{ml} \leftarrow f_3^{ml}$.

$P_1^1 =$	$J_5 : -1.49$	$J_4 : -1.38$	$J_2 : -1.27$	$J_3 : 0.3$	$J_1 : 0.72$	$f_1^{ml} = 523$
$P_2^1 =$	$J_4 : -1.92$	$J_2 : -1.59$	$J_1 : -0.98$	$J_3 : -0.05$	$J_5 : 0.65$	$f_2^{ml} = 523$
$P_3^1 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$	$f_3^{ml} = 495$

Atualização do melhor global

Tendo sido os valores de G^0 e f^{mg} definidos na iteração 0 , têm-se que:

$f_k^1 \leftarrow \min(f_1^{ml}, f_2^{ml}, f_3^{ml}) = \min(523, 523, 495) = 495$

$k \leftarrow 3$

Como $f_k^1 \geq f^{mg}$, então $G^1 \leftarrow G^0$ e $f^{mg} \leftarrow f^{mg}$.

$G^1 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$	$f^{mg} = 495$

Mutação

$z \leftarrow$ aleatório$(0\ ,\ 1) = 0.0$

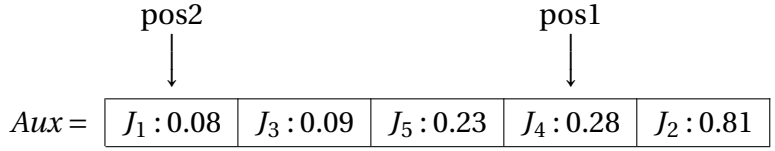
Como $z < Prob_m$, então deve-se aplicar a mutação em G^1 utilizando o operador mudança por deslocamento.

Seguindo o exemplo da figura 6 têm-se:

$pos1 \leftarrow$ aleatório$(1, 5) = 4$

$pos2 \leftarrow$ aleatório$(1, 5) = 1$

$Aux \leftarrow G^1$



Ao remover o trabalho J_4 da posição *pos1* e o inserir na posição *pos2*, os trabalhos J_1, J_3 e J_5 foram deslocados para a direita, no entanto os valores relacionados a cada posição não sofreram o deslocamento uma vez que deve-se sempre manter os valores de cada posição ordenados de forma crescente. Por fim obtém-se um novo *Aux* depois da mutação:

$$Aux = \begin{array}{|c|c|c|c|c|} \hline J_4:0.08 & J_1:0.09 & J_3:0.23 & J_5:0.28 & J_2:0.81 \\ \hline \end{array} \qquad C_{max} = 552$$

Como $C_{max}(Aux) \geq f^{mg}$ então não houve alteração no melhor global com a mutação, por este motivo G^1 e P_3^1 continuam os mesmos.

Critério de parada

Como $t < N_{sub}$, isto é, $1 < 3$ então o critério de parada não foi atingido e o processo continuará na próxima subida a partir da etapa de atualização da iteração.

2° Subida

Uma vez que os cálculos estão demonstrados em cada etapa do processo de subida na iteração anterior, as próximas subidas contém somente os valores para cada etapa.

Atualização da iteração

$t = 2$

Atualização do peso de inércia

$w^2 = 0.85$

Atualização das velocidades

$$\begin{array}{rc} V_1^2 = & \begin{array}{|c|c|c|c|c|} \hline 1.98 & 0.0 & 2.24 & -1.83 & -2.01 \\ \hline \end{array} \\ V_2^2 = & \begin{array}{|c|c|c|c|c|} \hline 0.86 & -0.07 & 0.9 & -1.4 & -0.01 \\ \hline \end{array} \\ V_3^2 = & \begin{array}{|c|c|c|c|c|} \hline -0.51 & 1.09 & -0.29 & -2.72 & -1.34 \\ \hline \end{array} \end{array}$$

Atualização das posições

$$\begin{array}{rc} X_1^2 = & \begin{array}{|c|c|c|c|c|} \hline J_5:0.49 & J_4:-1.38 & J_2:0.97 & J_3:-1.53 & J_1:-1.28 \\ \hline \end{array} \\ X_2^2 = & \begin{array}{|c|c|c|c|c|} \hline J_4:-1.06 & J_2:-1.66 & J_1:-0.07 & J_3:-1.45 & J_5:0.64 \\ \hline \end{array} \\ X_3^2 = & \begin{array}{|c|c|c|c|c|} \hline J_1:-0.43 & J_3:1.18 & J_5:-0.05 & J_4:-2.44 & J_2:-0.53 \\ \hline \end{array} \end{array}$$

Ordenando as posições

$X_1^2 =$	$J_3 : -1.53$	$J_4 : -1.38$	$J_1 : -1.28$	$J_5 : 0.49$	$J_2 : 0.97$	$C_{max} = 556$
$X_2^2 =$	$J_2 : -1.66$	$J_3 : -1.45$	$J_4 : -1.06$	$J_1 : -0.07$	$J_5 : 0.64$	$C_{max} = 570$
$X_3^2 =$	$J_4 : -2.44$	$J_2 : -0.53$	$J_1 : -0.43$	$J_5 : -0.05$	$J_3 : 1.18$	$C_{max} = 541$

Atualização dos melhores locais

$P_1^2 =$	$J_5 : -1.49$	$J_4 : -1.38$	$J_2 : -1.27$	$J_3 : 0.3$	$J_1 : 0.72$	$f_1^{ml} = 523$
$P_2^2 =$	$J_4 : -1.92$	$J_2 : -1.59$	$J_1 : -0.98$	$J_3 : -0.05$	$J_5 : 0.65$	$f_2^{ml} = 523$
$P_3^2 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$	$f_3^{ml} = 495$

Atualização do melhor global

$G^2 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$	$f^{mg} = 495$

Mutação

$z \leftarrow \text{aleatório}(0\ ,\ 1) = 0.42$

Como $z \geq Prob_m$, então não ocorreu a mutação e portanto P_i^2 e G^2 continuam os mesmos.

Critério de parada

Como $t < N_{sub}$, isto é, $2 < 3$ então o critério de parada não foi atingido e o processo continuará na próxima subida a partir da etapa de atualização da iteração.

3° Subida

Atualização da iteração

$t = 3$

Atualização do peso de inércia

$w^3 = 0.83$

Atualização das velocidades

$V_1^3 =$	2.27	2.56	4.0	−1.57	−1.57
$V_2^3 =$	3.0	1.13	2.52	−0.58	0.1
$V_3^3 =$	−0.43	0.92	−0.24	−2.31	−1.13

Atualização das posições

$X_1^3 =$	$J_5 : 0.78$	$J_4 : 1.18$	$J_2 : 2.73$	$J_3 : -1.27$	$J_1 : -0.85$
$X_2^3 =$	$J_4 : 1.08$	$J_2 : -0.46$	$J_1 : 1.54$	$J_3 : -0.63$	$J_5 : 0.75$
$X_3^3 =$	$J_1 : -0.35$	$J_3 : 1.01$	$J_5 : 0.0$	$J_4 : -2.03$	$J_2 : -0.31$

Ordenando as posições

$X_1^3 =$	$J_3 : -1.27$	$J_1 : -0.85$	$J_5 : 0.78$	$J_4 : 1.18$	$J_2 : 2.73$	$C_{max} = 497$
$X_2^3 =$	$J_3 : -0.63$	$J_2 : -0.46$	$J_5 : 0.75$	$J_4 : 1.08$	$J_1 : 1.54$	$C_{max} = 532$
$X_3^3 =$	$J_4 : -2.03$	$J_1 : -0.35$	$J_2 : -0.31$	$J_5 : 0.0$	$J_3 : 1.01$	$C_{max} = 552$

Atualização dos melhores locais

$P_1^3 =$	$J_3 : -1.27$	$J_1 : -0.85$	$J_5 : 0.78$	$J_4 : 1.18$	$J_2 : 2.73$	$f_1^{ml} = 497$
$P_2^3 =$	$J_4 : -1.92$	$J_2 : -1.59$	$J_1 : -0.98$	$J_3 : -0.05$	$J_5 : 0.65$	$f_2^{ml} = 523$
$P_3^3 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$	$f_3^{ml} = 495$

Atualização do melhor global

$G^3 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$	$f^{mg} = 495$

Mutação

$z \leftarrow \text{aleatório}(0\ ,\ 1) = 0.77$

Como $z \geq Prob_m$, então não ocorreu a mutação e portanto P_i^3 e G^3 continuam os mesmos.

Critério de parada

Como $t \geq N_{sub}$, isto é, $3 \geq 3$ então o critério de parada foi atingido, logo $X_i \leftarrow P_i^3$ e o processo de subida será interrompido. O retorno para a próxima etapa do BMH que é o processo observa-salta foi X_i:

$X_1 =$	$J_3 : -1.27$	$J_1 : -0.85$	$J_5 : 0.78$	$J_4 : 1.18$	$J_2 : 2.73$	$C_{max} = 497$
$X_2 =$	$J_4 : -1.92$	$J_2 : -1.59$	$J_1 : -0.98$	$J_3 : -0.05$	$J_5 : 0.65$	$C_{max} = 523$
$X_3 =$	$J_1 : 0.08$	$J_3 : 0.09$	$J_5 : 0.23$	$J_4 : 0.28$	$J_2 : 0.81$	$C_{max} = 495$

4.3.8.3 Processo observa-salta

Aqui será demonstrado o processo observa-salta.

Macaco M_1

Como não ocorreu nenhuma alteração no macaco M_1 para $z = 0$ e $z = 1$ o processo foi demonstrado a partir de $z = 2$.

A demonstração da atualização da posição será feita somente para y_1 pois os cálculos das demais posições de y são feitos de forma semelhante.

$y_1 = \text{aleatório}\big(-1.27 - 0.5,\ -1.27 + 0.5\big)$

$y_1 = \text{aleatório}\big(-1.57,\ -0.77\big)$

$y_1 = -1.14$

Análogo ao exemplo acima, escolhendo as demais posições obtém-se:

$y =$	-1.14	-1.33	0.88	0.96	2.29

$y \leftarrow y$ ordenado com MVP.

$y =$	$J_1 : -1.33$	$J_3 : -1.14$	$J_5 : 0.88$	$J_4 : 0.96$	$J_2 : 2.29$	$C_{max} = 495$

Como $C_{max}(y) \leq C_{max}(X_1)$, isto é, $495 \leq 497$, então:

$X_1 \leftarrow$	$J_1 : -1.33$	$J_3 : -1.14$	$J_5 : 0.88$	$J_4 : 0.96$	$J_2 : 2.29$

Macaco M_2

$z \leftarrow 0$

$$y = \begin{array}{|c|c|c|c|c|} \hline -1.98 & -1.62 & -1.33 & 0.32 & 0.33 \\ \hline \end{array}$$

$y \leftarrow y$ ordenado com MVP.

$$y = \begin{array}{|c|c|c|c|c|} \hline J_4 : -1.98 & J_2 : -1.62 & J_1 : -1.33 & J_3 : 0.32 & J_5 : 0.33 \\ \hline \end{array} \quad C_{max} = 523$$

Como $C_{max}(y) \leq C_{max}(X_2)$, isto é, $523 \leq 523$, então:

$$X_2 \leftarrow \begin{array}{|c|c|c|c|c|} \hline J_4 : -1.98 & J_2 : -1.62 & J_1 : -1.33 & J_3 : 0.32 & J_5 : 0.33 \\ \hline \end{array}$$

Macaco M_3

Para o macaco M_3 o número máximo de observações W_{obs} foi atingido e não houve alteração no valor de X_3, portanto não foi feita a demonstração para este caso. Com isto temos que X_3 foi mantido o mesmo.

Como o processo observa-salta já realizado para todos os macacos, então deve-se passar para a próxima etapa do BMH que é novamente o processo de subida.

4.3.8.4 Processo de subida II

Mais uma vez foi executado o processo de subida, tendo desta vez como população inicial os valores de X_i retornados do processo observa-salta.

Como não houve alterações durante todo o processo de subida ele não será demonstrado aqui sendo que a saída final deste processo foi:

$$\begin{array}{rc} X_1 = & \begin{array}{|c|c|c|c|c|} \hline J_1 : -1.33 & J_3 : -1.14 & J_5 : 0.88 & J_4 : 0.96 & J_2 : 2.29 \\ \hline \end{array} \quad C_{max} = 495 \\ X_2 = & \begin{array}{|c|c|c|c|c|} \hline J_4 : -1.98 & J_2 : -1.62 & J_1 : -1.33 & J_3 : 0.32 & J_5 : 0.33 \\ \hline \end{array} \quad C_{max} = 523 \\ X_3 = & \begin{array}{|c|c|c|c|c|} \hline J_1 : 0.08 & J_3 : 0.09 & J_5 : 0.23 & J_4 : 0.28 & J_2 : 0.81 \\ \hline \end{array} \quad C_{max} = 495 \end{array}$$

4.3.8.5 Processo salto mortal

A demonstração dos cálculos do pivô e da nova posição do macaco foram feitas somente para p_1 e $y_1 1$ respectivamente. Os cálculos das demais posições, tanto de p quanto de y são feitos de forma semelhante aos apresentados a seguir.

Para o pivô seguindo a equação 4.9 tem-se:

$p_1 = \big(-1.33 - 1.98 + 0.08\big) \div 3$

$p_1 = -3.23 \div 3$

$p_1 = -1.07$

Análogo ao exemplo acima, calculando os demais pivôs obtém-se:

$p =$					
	-1.07	-0.89	-0.07	0.52	1.14

Para calcular as novas posições deve-se primeiro escolher um valor de α:

$\alpha = \text{aleatório}\big(-1, 1\big) = -0.38$

e depois seguindo a equação 4.8 tem-se:

$y_1 1 = -1.33 + \big[-0.38\ \big(-1.07 + 1.33\big)\big]$

$y_1 1 = -1.33 - 0,09$

$y_1 1 = -1.42$

Análogo ao exemplo acima, determinando as novas posições obtém-se:

$y_1 =$	-1.42	-1.23	1.24	1.12	2.72
$y_2 =$	-1.6	-1.32	-0.81	0.4	0.66
$y_3 =$	0.17	0.16	0.25	0.26	0.78

Aplicando a regra do MVP nos valores de y tem-se:

$y_1 =$	$J_1 : -1.42$	$J_3 : -1.23$	$J_4 : 1.12$	$J_5 : 1.24$	$J_2 : 2.72$
$y_2 =$	$J_4 : -1.6$	$J_2 : -1.32$	$J_1 : -0.81$	$J_3 : 0.4$	$J_5 : 0.66$
$y_3 =$	$J_3 : 0.16$	$J_1 : 0.17$	$J_5 : 0.25$	$J_4 : 0.26$	$J_2 : 0.78$

Por fim temos que:

$X_i \leftarrow y_i$

4.3.8.6 Critério de parada

Como $iter < N_{iter}$, ou seja, $1 < 2$ então:

$iter \leftarrow iter + 1 = 2$

2° Ciclo de repetição do BMH

Como *iter* > 1, então a população inicial é composta pela população final (retorno do processo salto-mortal) do ciclo de repetição anterior, isto é, 1° ciclo de repetição.

1° Subida

Inicialização

$t \leftarrow 0$

Velocidade inicial

$V_1^0 =$	-1.47	0.9	3.15	-3.72	1.8
$V_2^0 =$	3.13	-0.74	1.13	1.27	0.32
$V_3^0 =$	-1.04	-2.61	-2.05	2.75	-1.16

População Inicial

$X_1^0 =$	$J_1 : -1.42$	$J_3 : -1.23$	$J_4 : 1.12$	$J_5 : 1.24$	$J_2 : 2.72$
$X_2^0 =$	$J_4 : -1.6$	$J_2 : -1.32$	$J_1 : -0.81$	$J_3 : 0.4$	$J_5 : 0.66$
$X_3^0 =$	$J_3 : 0.16$	$J_1 : 0.17$	$J_5 : 0.25$	$J_4 : 0.26$	$J_2 : 0.78$

Melhor local

$P_1^0 =$	$J_1 : -1.42$	$J_3 : -1.23$	$J_4 : 1.12$	$J_5 : 1.24$	$J_2 : 2.72$	$f_1^{ml} = 535$
$P_2^0 =$	$J_4 : -1.6$	$J_2 : -1.32$	$J_1 : -0.81$	$J_3 : 0.4$	$J_5 : 0.66$	$f_2^{ml} = 523$
$P_3^0 =$	$J_3 : 0.16$	$J_1 : 0.17$	$J_5 : 0.25$	$J_4 : 0.26$	$J_2 : 0.78$	$f_3^{ml} = 497$

Melhor global

$G^0 =$	$J_3 : 0.16$	$J_1 : 0.17$	$J_5 : 0.25$	$J_4 : 0.26$	$J_2 : 0.78$	$f^{mg} = 497$

Não será mostrada a execução deste ciclo de repetição pois ele é semelhante ao 1° ciclo de repetição e não houve melhora no valor do melhor global.

Critério de parada

Como $iter \geq N_{iter}$, ou seja, $2 \geq 2$ então o critério de parada do algoritmo foi atingido e a execução deste foi encerrada.

Portanto a saída final do algoritmo após a execução de todas os processos e ciclos de repetição foi:

J_1	J_3	J_5	J_4	J_2

$\boldsymbol{C_{max} = 495}$

5 Desenvolvimento

Neste capítulo está descrito o ambiente em que foram executadas as instâncias de Taillard para realizar os experimentos em cada um dos algoritmos propostos para o PFSP e também as instâncias que foram utilizadas nesta execução. Os resultados dos experimentos computacionais feitos nos algoritmos estudados nesta monografia foram analisados de modo que foi realizada uma comparação entre os resultados obtidos a partir da implementação dos algoritmos IG_{BLSP} e BMH com os resultados encontrados nos seus respectivos artigos originais. Posteriormente foi feita uma comparação entre o IG_{BLSP} e o BMH com a finalidade de saber qual método apresenta um melhor custo-benefício em relação a qualidade da solução encontrada e o tempo de execução de cada um.

A linguagem funcional *Clojure* [1] foi escolhida para o desenvolvimento de todos os algoritmos aqui analisados pelo fato de ser uma linguagem de programação robusta e prática, com um conjunto de recursos úteis que juntos formam uma ferramenta simples e coerente. O fato do *Clojure* rodar em cima da Máquina Virtual Java [2] (JVM) também foi um fator decisivo para a sua escolha, uma vez que a JVM é altamente otimizada e possui varias ferramentas para a realização de monitoramentos no código proporcionando melhorias e consequentemente ganho em desempenho.

Todo o código fonte resultante da implementação dos algoritmos aqui estudados bem como todos os resultados obtidos a partir da execução dos mesmos estão disponíveis publicamente em <https://github.com/ViniciusTxr/algoritmos-flowshop>. Os códigos fontes também podem ser encontrados nos Apêndices B, C e D deste trabalho.

5.1 Ambiente de execução

Todos os experimentos foram realizados em uma máquina virtual na nuvem da Microsoft Azure [3]. Foi escolhida uma máquina virtual da série Fsv2 pelo fato de ser da categoria de computação otimizada. A série F contêm a tecnologia *Hyper-Threading* e é baseada no processador Intel Xeon® Platinum 8168 (*SkyLake*) de 2,7 GHz, que pode alcançar velocidades de clock de até 3,7 GHz com a Tecnologia Intel Turbo Boost 2.0. De maneira mais específica, foi montada uma máquina com a instância F2s v2 com 2 vCPU's, 4GB de memória RAM e rodando o sistema operacional Ubuntu Server 16.04 LTS. Para o desenvolvimento foi utilizada a versão 1.8.0 do Clojure.

[1] Disponível em: <https://clojure.org/>
[2] Tradução direta do inglês de *Java Virtual Machine*
[3] Disponível em: <https://azure.microsoft.com>

5.2 Instâncias para análise

Todos os experimentos foram feitos baseados no *benchmark* proposto por Taillard (1993) com a finalidade de se chegar a resultados mais fiéis aos artigos originais, pelo fato de ele ter sido utilizado nas experiências computacionais tanto no algoritmo BMH proposto por Marichelvam, Tosun e Geetha (2017) quanto no algoritmo IG_{BLSP} proposto por Dubois-Lacoste, Pagnozzi e Stützle (2017). O fato do *benchmark* proposto por Taillard ser o mais utilizado na literatura recente pelos pesquisadores da área do *flow shop*, como mostram os dados encontrados na Tabela 7 contida no Capítulo 3, também fundamentam a escolha deste *benchmark* para a realização dos testes aqui encontrados.

Taillard propôs 12 conjuntos de testes para o problema *flow shop* com o objetivo de minimizar o *makespan*, sendo que cada um destes conjuntos tem um total de 10 instâncias. Isto quer dizer que no total tem-se 120 instâncias para a realização dos experimentos.

Os conjuntos são apresentados na forma *MxN*, onde *M* representa o número de máquinas e *N* representa o número de trabalhos de cada conjunto. Todos os 12 conjuntos propostos por Taillard foram utilizados em todos os algoritmos aqui estudados. Os conjuntos são: 20x5, 20x10 20x20, 50x5, 50x10, 50x20, 100x10, 100x20, 200x10, 200x20, 500x20, onde cada conjunto apresenta um nível de dificuldade diferente sendo que o mais simples é o 20x5, a complexidade vai subindo de acordo com a ordem em que os conjuntos foram apresentados acima e o mais difícil é o 500x20.

As instâncias serão representadas pelo nome *taix*, onde *x* é um inteiro que vai de 1 a 120. Com isto temos que a primeira instância do primeiro conjunto (20x5) receberá o nome *tai1*, a segunda instância do primeiro conjunto será *tai2*, e assim por diante até a última instância do último conjunto (500x20), que será chamada de *tai120*.

Cada instância tem um limitante inferior conhecido como *Lower Bound* (LB) e um limitante superior conhecido como *Upper Bound* (UB). O LB indica o menor valor que o *makespan* ótimo pode atingir, ou seja, o *makespan* ótimo vai ser sempre maior ou igual ao LB. Já o UB indica o melhor valor encontrado pelo algoritmo Busca Tabu implementado por Taillard e portanto é o valor máximo que o *makespan* ótimo atingirá, ou seja, o *makespan* ótimo vai ser sempre menor ou igual a UB. Uma descrição detalhada de como as instâncias foram criadas, do cálculo do LB, das sementes utilizadas para encontrar o UB e outras informações importantes sobre as instâncias estão documentadas em Taillard (1993).

5.3 Experimentos computacionais

Esta seção está dividida em três partes. Primeiramente foi feita uma comparação dos resultados obtidos através da implementação da heurística IG_{BLSP} feita aqui neste trabalho com os dados obtidos através do artigo original. Em um segundo momento foi realizada uma

comparação entre os dados coletados da meta-heurística BMH desenvolvida neste trabalho com os resultados obtidos no artigo original deste método. Em terceiro lugar os dados coletados de ambos algoritmos foram confrontados entre si para saber qual método apresenta uma melhor qualidade da solução em relação ao tempo de execução.

Cada uma das 120 instâncias foram executadas 10 vezes em cada algoritmo para uma análise mais justa e fidedigna dos dados obtidos. Durante a execução dos experimentos, para cada instância, foram coletados os seguintes dados: valor obtido a partir da heurística inicial NEH; melhor valor de *makespan*, pior valor de *makespan* e a média dos valores de *makespan* encontrados nas 10 execuções; tempo médio de processamento de cada instância.

5.3.1 Iteração gulosa com busca local em soluções parciais

Esta seção tem como objetivo verificar se a implementação realizada neste trabalho da heurística proposta por Dubois-Lacoste, Pagnozzi e Stützle (2017) pode chegar nos mesmos resultados do artigo original.

O algoritmo IG_{BLSP} proposto por Dubois-Lacoste, Pagnozzi e Stützle (2017) tem como critério de parada o tempo de processamento. Para a realização de seus experimentos, os autores utilizaram três limitantes diferentes para o cálculo dos tempos de processamento de parada. Esses três limitantes também foram utilizados neste trabalho e podem ser obtidos através da equação:

$$\frac{n \cdot m \cdot T}{2} \tag{5.1}$$

onde n representa o número de trabalhos da instância, m representa o número de máquinas da instância e T representa um tempo de CPU em milissegundos e pode receber os valores de 60, 120 e 240, formando assim os três limitantes para o tempo de processamento.

No artigo original o autor apresenta seus resultados a partir da média do desvio relativo percentual (DRP) de cada conjunto de instâncias para cada um dos três critérios de parada do algoritmo. Com o objetivo de se obter uma comparação fiel com o artigo original, os resultados obtidos a partir da implementação desta monografia que estão dispostos nesta seção também foram apresentados conforme a média do DRP.

O DRP, representado pelo simbolo σ, pode ser calculado através da equação:

$$\sigma = \frac{MSE - UB}{UB} \cdot 100 \tag{5.2}$$

onde *MSE* representa a melhor solução encontrada pelo algoritmo e *UB* significa o valor do *Upper Bound* descrito em Taillard (1993), ambos para a instância atualmente analisada.

Já a média do DRP para cada conjunto de instâncias, representada por ε, pode ser calculada da seguinte maneira:

$$\varepsilon = \frac{1}{10} \cdot \sum_{i=1}^{10} \sigma_i \tag{5.3}$$

Pelo fato do autor não apresentar os tempos de processamento no artigo original, os tempos de processamento obtidos a partir da implementação deste trabalho também não foram utilizados nesta seção, mas estão descritos na seção 5.3.3.

A tabela 17 apresenta os resultados de ε encontrados no artigo proposto por Dubois-Lacoste, Pagnozzi e Stützle (2017) (IG_{aut}) e os valores de ε encontrados a partir da implementação realizada neste trabalho (IG_{im}). Esta tabela mostra o resultante para os três limitantes do tempo de processamento. Os valores em negrito indicam o melhor valor do DRP entre as duas implementações em cada conjunto de instâncias. Já os valores marcados com asterisco indicam que o DRP foi melhor que o UB.

Tabela 17 – Comparação dos resultados obtidos em Dubois-Lacoste, Pagnozzi e Stützle (2017) com a implementação realizada neste trabalho da média do Desvio Relativo Percentual do algoritmo IG_{BLSP}

	T60		**T120**		**T240**	
Conjuntos	IG_{aut}	IG_{im}	IG_{aut}	IG_{im}	IG_{aut}	IG_{im}
20 x 5	0.0348	**-0.0080***	0.0308	**-0.0080***	0.0240	**-0.0080***
20 x 10	**0.0139**	0.0500	**0.0022**	0.0252	0	**-0.0072***
20 x 20	**0.0007**	0.0217	**0**	0.0392	**0**	0.0173
50 x 5	**0**	0.0141	**0**	0.0148	**0**	0.0078
50 x 10	**0.3887**	0.5579	0.3474	**0.3351**	0.3184	**0.2244**
50 x 20	**0.5358**	1.5628	**0.4417**	1.1461	**0.3703**	0.9061
100 x 5	**0**	0.0054	0	**-0.0377***	0	**-0.0457***
100 x 10	**0.0530**	0.5156	**0.0316**	0.4149	**0.0230**	0.2727
100 x 20	**0.8540**	1.5749	**0.7207**	1.2225	**0.6019**	0.9476
200 x 10	**0.0375**	0.4765	**0.0345**	0.4103	**0.0314**	0.2673
200 x 20	**0.8384**	1.7718	**0.7327**	1.5458	**0.6579**	1.1837
500 x 20	**0.3613**	0.9854	**0.3212**	0.9993	**0.2866**	0.9202
média	**0.2598**	0.6273	**0.2219**	0.5089	**0.1928**	0.3905

De acordo com os dados representados na Tabela 17 o algoritmo do autor teve um desempenho superior ao desenvolvido neste trabalho. Para as execuções cujo critério de parada foi *T60*, o algoritmo original foi melhor em 11 conjuntos de instâncias dos 12 possíveis, isto é, em 91,67% dos casos, conforme indicado nos valores em negrito. Somente no primeiro conjunto de instâncias (20x5), indicado com um asterisco, que o método desenvolvido neste trabalho se saiu melhor, onde na média foi melhor até mesmo que o *UB*.

Já no critério de parada *T120*, os resultados obtidos pelo autor foram melhores em 75% dos casos, ou seja, em 9 dos 12 conjuntos de instâncias analisados, como pode ser observado nos valores em negrito. Dos 3 conjuntos em que o algoritmo aqui desenvolvido foi superior, em 2 deles chegou-se a uma média do *makespan* melhor que o *Upper Bound*, conforme indicado nos valores marcados com um asterisco.

No terceiro e último limitante para o critério de parada, isto é, *T240*, o algoritmo original obteve um melhor resultado em 8 dos 12 conjuntos de instâncias representando 66,67% quando comparado ao algoritmo desenvolvido nesta monografia que foi melhor em 4 conjuntos. Destes 4 conjuntos, os resultados foram melhores em 3 deles em relação ao *Upper Bound*, como pode ser observado nos valores marcados com um asterisco.

As médias dos resultados encontrados em cada limitante mostra que o algoritmo aqui implementado está próximo do algoritmo original, uma vez que para *T60, T120 e T240* o algoritmo original teve respectivamente um DRP de 0,25%, 0,22% e 0,19% enquanto a implementação aqui realizada teve respectivamente uma média de DRP de 0,62%, 0,50% e 0,39%.

As médias mostram também que o melhor limitante utilizado como critério de parada para o algoritmo IG_{BLSP} foi o *T240*, pois nas duas implementações os melhores valores de *makespan* foram encontrados neste limitante.

Perante todos estes resultados tem-se que a implementação aqui realizada conseguiu chegar a valores de *makespan* bem próximos do artigo original e em alguns casos até mesmo melhorando a qualidade da solução quando comparado ao *Upper Bound*.

5.3.2 Busca do macaco híbrida

Esta seção tem como objetivo verificar se a implementação realizada neste trabalho da meta-heurística proposta por Marichelvam, Tosun e Geetha (2017) pode chegar nos mesmos resultados do artigo original.

No decorrer da seção estão dispostos os resultados encontrados no artigo original e os resultados obtidos através da implementação do mesmo nesta monografia. Pelo fato do autor não apresentar os tempos de processamento no artigo original, os tempos de processamento obtidos a partir da implementação deste trabalho também não foram utilizados nesta seção, mas estão descritos na seção 5.3.3.

O autor disponibilizou em seu artigo original os valores de *makespan* encontrados em cada uma das 120 instâncias propostas por Taillard. A Tabela 18 coloca lado a lado estes valores com o resultado da implementação realizada neste trabalho para cada instância, onde BMH_{aut} representa os resultados encontrados pelo autor do artigo original e BMH_{im} representa os melhores resultados encontrados a partir das 10 execuções da implementação realizada neste trabalho. Os valores em negrito indicam o melhor *makespan* encontrado entre as duas implementações para uma determinada instância.

Tabela 18 – Valores de *makespan* obtidos no artigo original proposto por Marichelvam, Tosun e Geetha (2017) e valores de *makespan* obtidos através da implementação realizada neste trabalho para o algoritmo BMH.

Instância	BMH_{aut}	BMH_{im}	Instância	BMH_{aut}	BMH_{im}
tai1	1282	**1278**	tai61	5502	**5493**
tai2	1362	**1359**	tai62	5272	**5268**
tai3	1094	**1083**	tai63	**5192**	5193
tai4	1304	**1293**	tai64	**5020**	5021
tai5	1248	**1235**	tai65	5254	**5253**
tai6	1204	**1195**	tai66	5144	**5137**
tai7	1247	**1239**	tai67	5264	**5247**
tai8	1208	**1206**	tai68	5114	**5108**
tai9	1254	**1230**	tai69	5466	**5454**
tai10	1124	**1108**	tai70	5332	**5328**
tai11	1616	**1588**	tai71	**5792**	5802
tai12	1712	**1669**	tai72	**5368**	5409
tai13	1510	**1500**	tai73	**5694**	5752
tai14	1384	**1383**	tai74	**5826**	5952
tai15	1456	**1425**	tai75	**5514**	5572
tai16	1420	**1407**	tai76	**5324**	5346
tai17	1496	**1486**	tai77	**5628**	5683
tai18	1568	**1551**	tai78	**5664**	5705
tai19	1604	**1602**	tai79	**5912**	5940
tai20	1615	**1608**	tai80	**5898**	5903
tai21	2324	**2312**	tai81	**6306**	6496
tai22	**2112**	2116	tai82	**6278**	6449
tai23	2348	**2343**	tai83	**6404**	6532
tai24	2242	**2233**	tai84	**6184**	6519
tai25	2320	**2310**	tai85	**6422**	6595
tai26	2249	**2232**	tai86	**6526**	6647
tai27	2290	**2286**	tai87	**6412**	6548
tai28	2224	**2208**	tai88	**6516**	6666
tai29	2246	**2242**	tai89	**6424**	6564
tai30	2192	**2188**	tai90	**6496**	6675
tai31	2728	**2724**	tai91	**10932**	10933
tai32	2846	**2843**	tai92	10624	**10618**
tai33	2642	**2622**	tai93	**11006**	11034

(continuação)

Instância	BMH_{aut}	BMH_{im}	Instância	BMH_{aut}	BMH_{im}
tai34	2762	**2753**	tai94	11024	**10970**
tai35	2866	**2863**	tai95	**10474**	10614
tai36	2832	**2831**	tai96	**10369**	10424
tai37	2748	**2725**	tai97	**10907**	10954
tai38	2690	**2683**	tai98	**10794**	10846
tai39	2564	**2561**	tai99	**10482**	10579
tai40	2796	**2782**	tai100	**10720**	10760
tai41	**3034**	3096	tai101	**11342**	11535
tai42	2994	**2967**	tai102	**11584**	11700
tai43	**2924**	2941	tai103	**11568**	11772
tai44	**3082**	3114	tai104	**11480**	11727
tai45	**3028**	3071	tai105	**11452**	11661
tai46	**3064**	3113	tai106	**11378**	11623
tai47	**3132**	3184	tai107	**11624**	11806
tai48	**3072**	3085	tai108	**11613**	11767
tai49	**2924**	2984	tai109	**11524**	11604
tai50	**3134**	3175	tai110	**11574**	11742
tai51	**3896**	3987	tai111	**26552**	26637
tai52	**3746**	3859	tai112	**26823**	27122
tai53	**3694**	3811	tai113	**26627**	26893
tai54	**3814**	3886	tai114	**26674**	26948
tai55	**3686**	3744	tai115	**26622**	26767
tai56	**3722**	3832	tai116	**26722**	26934
tai57	**3766**	3839	tai117	**26572**	26722
tai58	**3768**	3873	tai118	**26720**	27139
tai59	**3812**	3875	tai119	**26210**	26560
tai60	**3826**	3927	tai120	**26720**	26956

A Tabela 19 mostra um comparativo com relação a média do DRP encontrada a partir dos resultados mostrados na Tabela 18. O DRP é calculado conforme a Equação 5.2 e a média do DRP é calculada através da Equação 5.3. Os valores em negrito indicam a melhor média do DRP encontrado entre a implementação do autor e a realizada neste trabalho para um determinado conjunto de instâncias.

Como visto na Tabela 18 os valores de *makespan* adquiridos através da implementação realizada neste trabalho estão próximos dos resultados do artigo original. Das 120 instâncias, em 50 delas (41,67%) a implementação realizada neste trabalho apresentou resulta-

dos melhores que a implementação original. Como em nenhuma instância o resultado das duas implementações foi igual, então consequentemente a implementação original foi melhor em 70 instâncias (58,33%) em relação a implementação deste trabalho.

Já a Tabela 19 nos mostra que a média do DRP em cada conjunto de instâncias da implementação realizada nesta monografia também foi próxima as médias do DRP obtidas a partir do artigo original. Nos conjuntos de instâncias tidos como mais fáceis, isto é, de menor complexidade, a implementação deste trabalho se saiu um pouco melhor, já nas instâncias mais complexas o algoritmo do autor obteve resultados melhores.

Tabela 19 – Comparação dos resultados obtidos em Marichelvam, Tosun e Geetha (2017) com a implementação realizada neste trabalho em relação a média do desvio relativo percentual do algoritmo BMH

Conjuntos	BMH_{aut}	BMH_{im}
20 x 5	0.8517	**0.0104**
20 x 10	1.5926	**0.5363**
20 x 20	0.8730	**0.5312**
50 x 5	0.4060	**0.0851**
50 x 10	**1.3882**	2.5201
50 x 20	**1.1271**	3.5483
100 x 5	0.1733	**0.0641**
100 x 10	**0.5660**	1.3547
100 x 20	**0.5347**	3.2464
200 x 10	**0.5610**	0.9406
200 x 20	**1.2323**	2.8140
500 x 20	**0.6608**	1.5812
média	**0.8305**	1.4360

A diferença entre os resultados encontrados pelo algoritmo original e os resultados encontrados a partir da implementação realizada nesta monografia pode ser justificada pelo fato dos autores do algoritmo original não revelarem alguns parâmetros importantes da implementação realizada por eles, com isto estes parâmetros tiveram que ser abstraídos e calculados nesta monografia, levando a não exatidão desta implementação quando comparada com a original. No entanto a partir dos resultados apresentados na Tabela 19 tem-se que o algoritmo BMH desenvolvido nesta monografia condiz com o proposto em Marichelvam, Tosun e Geetha (2017), uma vez que as diferenças entre o DRP das implementações foram pequenas.

5.3.3 Comparação entre o IG_{BLSP} e o BMH

Nesta seção tem-se uma comparação entre os dois métodos que foram implementados neste trabalho, o IG_{BLSP} e o BMH, com a finalidade de saber qual dois dois apresenta um

melhor custo-benefício quando levados em conta o tempo de processamento e a qualidade da solução encontrada.

Durante as 10 execuções de cada instância, foram coletadas as seguintes informações de ambos os algoritmos para uma comparação justa: o melhor *makespan* encontrado, o pior *makespan* encontrado, uma média dos *makespans* encontrados nas 10 execuções, a média de tempo das 10 execuções e o DRP do melhor encontrado em relação ao *Upper Bound*.

Os resultados finais com todos os dados obtidos a partir da execução de cada algoritmo implementado neste trabalho estão disponíveis publicamente e podem ser encontrados em <https://github.com/ViniciusTxr/algoritmos-flowshop>. No algoritmo IG_{BLSP} foi coletado um dado a mais, o número de iterações que foram executadas em cada uma 10 execuções de cada instância, no entanto este dado não será usado para comparação com o método BMH. Este dado foi coletado apenas para fins acadêmicos caso algum outro pesquisador necessite para uma comparação futura.

A Tabela 20 apresenta os resultados obtidos a partir da execução da heurística IG_{BLSP}. Já a Tabela 21 apresenta os resultados encontrados a partir da execução da meta-heurística BMH. Os valores contidos na coluna 'NEH' das duas tabelas representa o *makespan* encontrado a partir da heurística inicial NEH implementada tanto no IG_{BLSP} quanto no BMH. Já a coluna 'Iterações' presente na Tabela 20 representa a media de iterações realizadas durante as 10 execuções do algoritmo IG_{BLSP}.

Tabela 20 – Dados coletados a partir da implementação realizada neste trabalho para o algoritmo IG_{BLSP}

Instância	NEH	*UB*	Melhor	Pior	Média	Tempo (s)	DRP (%)	Iterações
tai1	1286	1278	1278	1278	1278.0	12.01	0.0	924.5
tai2	1365	1359	1359	1359	1359.0	12.0	0.0	806.7
tai3	1140	1081	1081	1081	1081.0	12.01	0.0	745.6
tai4	1325	1293	1293	1297	1293.8	12.01	0.0	805.1
tai5	1305	1236	1235	1235	1235.0	12.0	-0.0809	851.0
tai6	1228	1195	1195	1195	1195.0	12.0	0.0	833.1
tai7	1278	1239	1239	1239	1239.0	12.0	0.0	877.7
tai8	1223	1206	1206	1206	1206.0	12.01	0.0	721.7
tai9	1291	1230	1230	1230	1230.0	12.0	0.0	827.3
tai10	1151	1108	1108	1108	1108.0	12.01	0.0	707.5
tai11	1680	1582	1582	1582	1582.0	24.01	0.0	1018.2
tai12	1786	1659	1659	1660	1659.3	24.01	0.0	1077.2
tai13	1557	1496	1496	1505	1499.1	24.02	0.0	827.4
tai14	1439	1378	1377	1384	1379.0	24.01	-0.0725	876.2

(continuação)

Instância	NEH	*UB*	Melhor	Pior	Média	Tempo (s)	DRP (%)	Iterações
tai15	1502	1419	1419	1422	1419.3	24.01	0.0	900.0
tai16	1453	1397	1397	1400	1397.3	24.01	0.0	865.3
tai17	1562	1484	1484	1484	1484.0	24.01	0.0	1045.5
tai18	1609	1538	1543	1550	1545.3	24.02	0.325	835.7
tai19	1647	1593	1593	1594	1593.1	24.0	0.0	857.1
tai20	1653	1591	1591	1604	1593.0	24.01	0.0	807.1
tai21	2410	2297	2297	2298	2297.2	48.02	0.0	966.0
tai22	2150	2100	2099	2101	2100.2	48.02	-0.0476	963.0
tai23	2429	2326	2326	2337	2331.4	48.02	0.0	944.7
tai24	2262	2223	2223	2224	2223.1	48.02	0.0	975.6
tai25	2397	2291	2294	2300	2295.6	48.03	0.1309	889.0
tai26	2349	2226	2228	2235	2229.4	48.02	0.0898	892.5
tai27	2362	2273	2273	2278	2274.9	48.03	0.0	1025.4
tai28	2249	2200	2200	2206	2201.2	48.02	0.0	1125.9
tai29	2320	2237	2237	2242	2240.5	48.02	0.0	1012.8
tai30	2277	2178	2178	2180	2179.0	48.03	0.0	910.5
tai31	2733	2724	2724	2724	2724.0	30.13	0.0	121.3
tai32	2882	2834	2834	2838	2837.4	30.1	0.0	129.3
tai33	2640	2621	2621	2621	2621.0	30.17	0.0	108.9
tai34	2785	2751	2751	2762	2752.7	30.17	0.0	111.5
tai35	2868	2863	2863	2864	2863.9	30.1	0.0	128.5
tai36	2838	2829	2829	2831	2829.9	30.12	0.0	135.4
tai37	2753	2725	2725	2725	2725.0	30.12	0.0	131.1
tai38	2694	2683	2683	2683	2683.0	30.1	0.0	127.5
tai39	2574	2552	2554	2561	2556.0	30.17	0.0783	104.3
tai40	2803	2782	2782	2782	2782.0	30.13	0.0	123.3
tai41	3168	3025	3025	3042	3035.9	60.22	0.0	139.3
tai42	3032	2892	2904	2923	2917.5	60.21	0.4149	120.7
tai43	2990	2864	2871	2886	2876.0	60.39	0.2444	114.7
tai44	3198	3064	3064	3088	3073.4	60.19	0.0	141.4
tai45	3128	2986	3008	3027	3014.9	60.25	0.7367	132.8
tai46	3172	3006	3010	3032	3020.4	60.25	0.133	140.3
tai47	3277	3107	3115	3140	3124.8	60.17	0.2574	148.0
tai48	3193	3039	3044	3058	3048.4	60.26	0.1645	113.5
tai49	3015	2902	2903	2917	2909.9	60.35	0.0344	114.8
tai50	3272	3091	3099	3125	3111.1	60.27	0.2588	132.2

(continuação)

Instância	NEH	*UB*	Melhor	Pior	Média	Tempo (s)	DRP (%)	Iterações
tai51	4038	3875	3910	3956	3929.9	120.45	0.9032	113.6
tai52	3921	3715	3746	3790	3762.2	120.72	0.8344	123.8
tai53	3923	3668	3695	3748	3721.3	120.53	0.736	154.6
tai54	3969	3752	3779	3823	3793.0	120.67	0.7196	119.1
tai55	3822	3635	3671	3700	3681.9	120.34	0.9903	124.7
tai56	3920	3698	3726	3787	3749.0	120.43	0.7571	144.2
tai57	3952	3716	3740	3812	3778.8	120.5	0.6458	115.5
tai58	3919	3709	3770	3815	3786.8	120.65	1.6446	120.3
tai59	3952	3765	3805	3825	3814.7	120.73	1.0624	110.3
tai60	4079	3777	3806	3843	3819.3	120.43	0.7678	114.4
tai61	5519	5493	5493	5493	5493.0	60.54	0.0	36.4
tai62	5348	5268	5268	5284	5278.8	60.61	0.0	38.3
tai63	5218	5175	5175	5179	5175.4	61.61	0.0	32.3
tai64	5023	5014	5014	5023	5018.4	60.66	0.0	35.3
tai65	5267	5250	5250	5253	5250.6	61.26	0.0	30.4
tai66	5139	5135	5135	5139	5136.6	61.03	0.0	45.4
tai67	5272	5246	5246	5256	5247.9	60.91	0.0	30.6
tai68	5129	5106	5094	5106	5096.4	61.21	-0.235	28.8
tai69	5489	5454	5448	5465	5453.0	60.69	-0.11	40.3
tai70	5338	5328	5322	5328	5323.2	61.38	-0.1126	35.4
tai71	5879	5770	5781	5803	5788.7	121.87	0.1906	37.4
tai72	5471	5349	5362	5391	5371.1	122.08	0.243	29.4
tai73	5806	5677	5679	5699	5690.6	122.34	0.0352	37.1
tai74	6004	5791	5834	5877	5848.5	122.08	0.7425	27.5
tai75	5650	5468	5507	5535	5517.1	122.26	0.7132	30.5
tai76	5429	5303	5308	5323	5318.9	121.99	0.0942	32.6
tai77	5725	5599	5599	5642	5625.3	122.81	0.0	31.8
tai78	5801	5623	5636	5694	5661.6	122.47	0.2311	31.9
tai79	5995	5875	5888	5938	5911.0	122.36	0.2212	35.4
tai80	5904	5845	5860	5903	5891.6	121.41	0.2566	38.4
tai81	6591	6286	6350	6386	6368.3	243.82	1.0181	26.1
tai82	6565	6241	6301	6339	6316.7	244.83	0.9613	31.8
tai83	6650	6329	6389	6432	6408.7	245.83	0.948	28.3
tai84	6562	6306	6388	6428	6402.0	244.77	1.3003	26.0
tai85	6645	6377	6409	6485	6457.0	244.02	0.5018	35.7
tai86	6692	6437	6491	6539	6514.9	245.09	0.8389	34.1

(continuação)

Instância	NEH	*UB*	Melhor	Pior	Média	Tempo (s)	DRP (%)	Iterações
tai87	6715	6346	6392	6454	6423.2	245.2	0.7248	24.9
tai88	6780	6481	6552	6585	6565.5	245.23	1.0955	31.1
tai89	6644	6358	6415	6452	6434.3	246.05	0.8965	28.0
tai90	6697	6465	6542	6576	6562.4	244.22	1.191	32.5
tai91	10942	10868	10872	10942	10897.0	249.81	0.0368	10.5
tai92	10679	10494	10529	10584	10558.4	251.59	0.3335	8.7
tai93	11081	10922	10965	11044	11014.5	250.71	0.3937	10.0
tai94	11057	10889	10939	11057	11012.0	245.99	0.4591	11.5
tai95	10648	10524	10537	10569	10542.9	254.78	0.1235	8.5
tai96	10467	10331	10348	10401	10379.5	256.96	0.1645	10.3
tai97	10965	10857	10882	10933	10899.4	249.05	0.2302	12.0
tai98	10857	10731	10752	10835	10785.1	257.14	0.1956	8.8
tai99	10628	10438	10465	10490	10477.8	259.87	0.2586	8.4
tai100	10760	10676	10727	10758	10738.0	254.77	0.4777	9.8
tai101	11624	11294	11420	11446	11435.1	506.8	1.1156	8.4
tai102	11789	11420	11465	11586	11522.3	541.77	0.394	5.3
tai103	11938	11446	11614	11690	11641.0	508.15	1.4677	6.0
tai104	11771	11347	11539	11587	11565.0	501.98	1.692	8.2
tai105	11738	11311	11462	11562	11521.0	530.25	1.3349	6.1
tai106	11762	11282	11414	11508	11461.2	529.17	1.17	6.0
tai107	11879	11456	11550	11635	11603.5	528.23	0.8205	6.7
tai108	11836	11415	11600	11667	11633.2	531.81	1.6206	6.4
tai109	11642	11343	11468	11534	11492.7	517.89	1.102	7.3
tai110	11755	11422	11550	11635	11597.0	515.91	1.1206	6.7
tai111	26670	26189	26444	26546	26514.7	2150.6	0.9736	1.2
tai112	27153	26629	26936	27053	27010.9	1753.58	1.1528	1.6
tai113	26894	26458	26706	26850	26761.8	1875.87	0.9373	1.6
tai114	26960	26549	26769	26866	26824.4	1771.01	0.8286	1.4
tai115	26808	26404	26613	26679	26644.5	1725.57	0.7915	1.3
tai116	27025	26581	26784	26900	26843.1	1740.8	0.7637	1.1
tai117	26749	26461	26609	26691	26661.8	1739.67	0.5593	1.4
tai118	27143	26615	26936	27034	26992.0	2011.2	1.206	1.4
tai119	26580	26083	26365	26498	26436.1	1686.44	1.0811	1.1
tai120	26969	26527	26768	26857	26810.4	1798.26	0.9085	1.3

Como visto na Tabela 20, a heurística IG_{BLSP} aqui implementada foi chamada de IG_{im}. Este método conseguiu um valor de *makespan* menor que o *UB* em 6 instâncias (tai5, tai14, tai22, tai68, tai69, tai70) representando 5% das 120. Em 43 instâncias, isto é, 35,84% de todo o conjunto, o *makespan* encontrado foi igual ao *UB*. Nas outras 71 instâncias (59,16%) o *makespan* foi maior que o *UB*.

Tabela 21 – Dados coletados a partir da implementação realizada neste trabalho para o algoritmo BMH

Instância	NEH	*UB*	Melhor	Pior	Média	Tempo (s)	DRP (%)
tai1	1286	1278	1278	1286	1279.2	738.76	0.0
tai2	1365	1359	1359	1359	1359.0	702.1	0.0
tai3	1140	1081	1083	1089	1087.8	669.36	0.185
tai4	1325	1293	1293	1301	1297.4	813.87	0.0
tai5	1305	1236	1235	1244	1242.2	709.71	-0.0809
tai6	1228	1195	1195	1195	1195.0	691.2	0.0
tai7	1278	1239	1239	1249	1240.0	694.28	0.0
tai8	1223	1206	1206	1206	1206.0	805.41	0.0
tai9	1291	1230	1230	1249	1236.9	697.44	0.0
tai10	1151	1108	1108	1111	1108.5	816.54	0.0
tai11	1680	1582	1588	1614	1601.6	1000.81	0.3792
tai12	1786	1659	1669	1692	1681.5	1049.76	0.6027
tai13	1557	1496	1500	1530	1518.9	1050.5	0.2673
tai14	1439	1378	1383	1402	1393.7	984.89	0.3628
tai15	1502	1419	1425	1441	1435.3	971.35	0.4228
tai16	1453	1397	1407	1424	1414.6	1088.69	0.7158
tai17	1562	1484	1486	1504	1494.2	1016.92	0.1347
tai18	1609	1538	1551	1572	1562.4	1068.2	0.8452
tai19	1647	1593	1602	1622	1611.1	944.11	0.5649
tai20	1653	1591	1608	1626	1616.5	1070.14	1.0685
tai21	2410	2297	2312	2339	2324.8	1389.04	0.653
tai22	2150	2100	2116	2136	2126.2	1473.57	0.7619
tai23	2429	2326	2343	2362	2355.4	1351.42	0.7308
tai24	2262	2223	2233	2250	2242.8	1403.43	0.4498
tai25	2397	2291	2310	2328	2317.9	1430.15	0.8293
tai26	2349	2226	2232	2258	2249.2	1364.61	0.2695
tai27	2362	2273	2286	2312	2300.3	1289.71	0.5719
tai28	2249	2200	2208	2241	2224.5	1470.67	0.3636
tai29	2320	2237	2242	2291	2265.1	1517.05	0.2235

(continuação)

Instância	NEH	*UB*	Melhor	Pior	Média	Tempo (s)	DRP (%)
tai30	2277	2178	2188	2213	2203.0	1531.11	0.4591
tai31	2733	2724	2724	2729	2724.5	1952.72	0.0
tai32	2882	2834	2843	2848	2846.8	2037.57	0.3175
tai33	2640	2621	2622	2629	2625.0	2193.5	0.0381
tai34	2785	2751	2753	2770	2763.5	2092.48	0.0727
tai35	2868	2863	2863	2864	2863.7	1892.39	0.0
tai36	2838	2829	2831	2834	2832.3	2178.36	0.0706
tai37	2753	2725	2725	2736	2731.4	2153.89	0.0
tai38	2694	2683	2683	2694	2687.7	2153.13	0.0
tai39	2574	2552	2561	2570	2566.1	2150.0	0.3526
tai40	2803	2782	2782	2783	2782.1	1942.07	0.0
tai41	3168	3025	3096	3148	3124.8	2566.64	2.3471
tai42	3032	2892	2967	3009	2986.9	2756.02	2.5933
tai43	2990	2864	2941	2985	2958.8	2695.51	2.6885
tai44	3198	3064	3114	3198	3142.4	2645.23	1.6318
tai45	3128	2986	3071	3123	3095.7	2682.49	2.8466
tai46	3172	3006	3113	3158	3139.6	2683.17	3.5595
tai47	3277	3107	3184	3256	3211.8	2698.83	2.4782
tai48	3193	3039	3085	3167	3120.0	2648.29	1.5136
tai49	3015	2902	2984	3015	2998.7	2706.0	2.8256
tai50	3272	3091	3175	3221	3201.9	2696.61	2.7175
tai51	4038	3875	3987	4038	4006.7	3665.34	2.8903
tai52	3921	3715	3859	3918	3883.4	3639.04	3.8761
tai53	3923	3668	3811	3897	3852.7	3607.12	3.8985
tai54	3969	3752	3886	3938	3907.0	3226.67	3.5714
tai55	3822	3635	3744	3822	3784.4	3534.85	2.9986
tai56	3920	3698	3832	3888	3857.6	3491.1	3.6235
tai57	3952	3716	3839	3915	3893.6	3615.54	3.31
tai58	3919	3709	3873	3915	3897.5	3680.89	4.4216
tai59	3952	3765	3875	3940	3915.7	3689.11	2.9216
tai60	4079	3777	3927	4013	3964.4	3635.7	3.9714
tai61	5519	5493	5493	5495	5493.6	3788.53	0.0
tai62	5348	5268	5268	5290	5283.0	3949.02	0.0
tai63	5218	5175	5193	5206	5200.6	4319.71	0.3478
tai64	5023	5014	5021	5023	5022.5	3956.97	0.1396
tai65	5267	5250	5253	5266	5256.9	4151.26	0.0571

(continuação)

Instância	NEH	*UB*	Melhor	Pior	Média	Tempo (s)	DRP (%)
tai66	5139	5135	5137	5139	5137.9	4012.04	0.0389
tai67	5272	5246	5247	5272	5258.4	5899.25	0.019
tai68	5129	5106	5108	5129	5122.6	4269.75	0.0391
tai69	5489	5454	5454	5486	5473.0	3958.14	0.0
tai70	5338	5328	5328	5337	5331.6	4270.41	0.0
tai71	5879	5770	5802	5879	5837.3	4907.74	0.5545
tai72	5471	5349	5409	5464	5452.6	4961.17	1.1217
tai73	5806	5677	5752	5800	5776.3	4639.9	1.3211
tai74	6004	5791	5952	5997	5974.3	4859.99	2.7801
tai75	5650	5468	5572	5636	5606.9	5294.23	1.9019
tai76	5429	5303	5346	5401	5376.3	4775.91	0.8108
tai77	5725	5599	5683	5725	5703.7	5103.77	1.5002
tai78	5801	5623	5705	5785	5739.8	6756.06	1.4582
tai79	5995	5875	5940	5995	5974.0	5068.8	1.1063
tai80	5904	5845	5903	5904	5903.2	4897.44	0.9923
tai81	6591	6286	6496	6591	6563.5	6806.17	3.3407
tai82	6565	6241	6449	6565	6512.8	6510.06	3.3327
tai83	6650	6329	6532	6650	6602.2	6255.75	3.2074
tai84	6562	6306	6519	6562	6548.4	6704.51	3.3777
tai85	6645	6377	6595	6645	6628.5	6671.29	3.4185
tai86	6692	6437	6647	6692	6668.1	6698.73	3.2623
tai87	6715	6346	6548	6663	6622.8	6669.67	3.1831
tai88	6780	6481	6666	6780	6726.9	5928.8	2.8544
tai89	6644	6358	6564	6644	6608.0	6046.06	3.24
tai90	6697	6465	6675	6697	6689.7	6887.2	3.2482
tai91	10942	10868	10933	10942	10941.1	9858.06	0.598
tai92	10679	10494	10618	10679	10660.7	10851.02	1.1816
tai93	11081	10922	11034	11066	11058.4	10188.51	1.0254
tai94	11057	10889	10970	11052	11030.8	10315.78	0.7438
tai95	10648	10524	10614	10648	10636.1	9902.0	0.8551
tai96	10467	10331	10424	10467	10454.3	9911.0	0.9002
tai97	10965	10857	10954	10965	10963.2	10800.62	0.8934
tai98	10857	10731	10846	10857	10854.9	10380.06	1.0716
tai99	10628	10438	10579	10628	10617.0	9959.73	1.3508
tai100	10760	10676	10760	10760	10760.0	10280.10	0.7868
tai101	11624	11294	11535	11624	11605.5	13684.71	2.1338

(continuação)

Instância	NEH	*UB*	Melhor	Pior	Média	Tempo (s)	DRP (%)
tai102	11789	11420	11700	11789	11767.8	12856.89	2.4518
tai103	11938	11446	11772	11928	11850.1	13377.53	2.8481
tai104	11771	11347	11727	11771	11752.9	14293.9	3.3489
tai105	11738	11311	11661	11738	11712.1	13531.94	3.0943
tai106	11762	11282	11623	11756	11701.5	13425.62	3.0225
tai107	11879	11456	11806	11879	11852.8	13865.55	3.0551
tai108	11836	11415	11767	11836	11807.2	14643.54	3.0836
tai109	11642	11343	11604	11642	11630.7	13224.45	2.3009
tai110	11755	11422	11742	11755	11751.5	13933.04	2.8016
tai111	26670	26189	26637	26670	26663.1	37418.02	1.7106
tai112	27153	26629	27122	27153	27141.7	37076.96	1.8513
tai113	26894	26458	26893	26894	26893.5	35515.88	1.6441
tai114	26960	26549	26948	26960	26957.7	36052.08	1.5028
tai115	26808	26404	26767	26808	26793.4	38279.51	1.3747
tai116	27025	26581	26934	27025	26994.4	35714.17	1.328
tai117	26749	26461	26722	26749	26743.5	35696.75	0.9863
tai118	27143	26615	27139	27143	27142.6	32724.68	1.9688
tai119	26580	26083	26560	26580	26576.6	36519.07	1.8287
tai120	26969	26527	26956	26969	26966.2	32644.83	1.6172

Já a meta-heurística BMH foi chamada de BMH_{im} na Tabela 21, encontrou apenas um valor de *makespan* abaixo do *UB* (tai5), ou seja 0,84% dos casos. Em 17 instâncias, ou seja 14,16% das vezes, o *makespan* foi igual ao *UB*. Nas outras 102 instâncias, isto é, 85% dos casos, o *makespan* encontrado pelo BMH_{im} foi pior que o *UB*.

Na Tabela 22 foi apresentada uma comparação da média dos valores do melhor *makespan*, do pior *makespan* e da média dos *makespans* encontrados em cada uma das instâncias entre os dois métodos para cada conjunto de instâncias. Em negrito estão representados os melhores valores encontrados em cada uma das comparações entre as duas implementações para cada conjunto de instâncias.

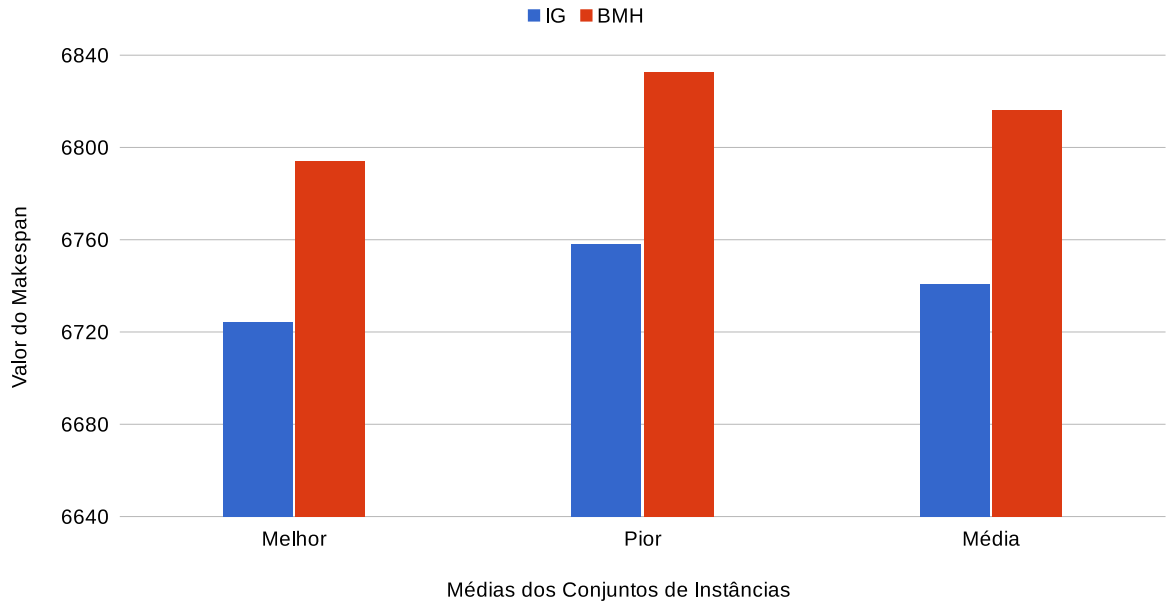
Na Tabela 22 é mostrado que a heurística IG_{BLSP} obteve um resultado melhor que o BMH em todas as comparações do melhor, pior e da média do *makespan* para todos os conjuntos de instância, ou seja, em 100% dos casos. A tabela ainda indica que em 75% das vezes a média do pior resultado encontrado pelo IG_{BLSP} foi superior a média do melhor encontrado pelo BMH, conforme pode ser observado nos valores marcados com um asterisco. A Figura 7 ajuda a representar estes resultados graficamente, onde tem-se os valores médios do melhor, pior e da média para todos os conjuntos de instâncias.

Tabela 22 – Comparação dos resultados do melhor, pior e a média dos *makespans* obtidos em cada conjunto de instâncias a partir das implementações do IG_{BLSP} e BMH

		Melhor		**Pior**		**Média**	
Conjunto	UB	IG_{im}	BMH_{im}	IG_{im}	BMH_{im}	IG_{im}	BMH_{im}
20 x 5	1222.5	**1222.4**	1222.6	**1222.8**	1228.9	**1222.4**	1225.1
20 x 10	1513.7	**1514.1**	1521.9*	**1518.5***	1542.7	**1515.1**	1532.9
20 x 20	2235.1	**2235.5**	2247.0*	**2240.1***	2273.0	**2237.2**	2260.9
50 x 5	2736.4	**2736.6**	2738.7	**2739.1**	2745.7	**2737.4**	2742.3
50 x 10	2997.6	**3004.3**	3073.0*	**3023.8***	3128.0	**3013.2**	3098.0
50 x 20	3731.0	**3764.8**	3863.3*	**3809.9***	3928.4	**3783.6**	3896.3
100 x 5	5246.9	**5244.5**	5250.2	**5252.6**	5264.3	**5247.3**	5258.0
100 x 10	5630.0	**5645.4**	5706.4*	**5680.5***	5758.6	**5662.4**	5734.4
100 x 20	6362.6	**6422.9**	6569.1*	**6467.6***	6648.9	**6445.3**	6617.0
200 x 10	10673.0	**10701.6**	10773.2*	**10761.3***	10806.4	**10730.4**	10797.6
200 x 20	11373.6	**11508.2**	11693.7*	**11585.0***	11771.8	**11547**.2	11743.2
500 x 20	26449.6	**26693.0**	26867.8*	**26797.4***	26895.1	**26749.9**	26887.2
média	6681.0	**6724.4**	6793.9	**6758.2**	6832.6	**6740.9**	6816.1

A Figura 7 ajuda a representar estes resultados graficamente, onde tem-se os valores médios do melhor, pior e da média para todos os conjuntos de instâncias.

Figura 7 – Comparação das médias do melhor, pior e da média dos *makespans* encontrados nos algoritmos IG_{im} e BMH_{im}.



Já na Tabela 23 foi detalhada a média, para cada conjunto de instâncias, dos valores do DRP em relação ao *Upper Bound*, o tempo de execução em segundos, e a porcentagem de melhoramento obtido a partir do melhor valor de *makespan* em relação ao valor encontrado pela heurística inicial NEH. Em negrito estão representados os melhores valores en-

contrados em cada uma das comparações entre os dois algoritmos para cada conjunto de instâncias. Mais uma vez o IG_{BLSP} se mostra superior ao BMH em todos os quesitos aqui comparados, isto é, em 100% dos casos.

Tabela 23 – Comparação dos resultados do DRP, tempo de execução e a porcentagem de melhoramento em relação ao NEH obtidos em cada conjunto de instâncias a partir das implementações do IG_{BLSP} e BMH

	DRP (%)		**Tempo (s)**		**Melhoramento NEH (%)**	
Conjunto	IG_{im}	BMH_{im}	IG_{im}	BMH_{im}	IG_{im}	BMH_{im}
20 x 5	**-0.0080***	0.0104	**12.00**	733.86	**2.9605**	2.9430
20 x 10	**-0.0072***	0.5363	**24.01**	1024.53	**4.6674**	4.1796
20 x 20	**0.0173**	0.5312	**48.02**	1422.07	**3.6345**	3.1391
50 x 5	**0.0078**	0.0851	**30.13**	2074.61	**0.7378**	0.6613
50 x 10	**0.2244**	2.5201	**60.25**	2677.87	**4.4460**	2.2576
50 x 20	**0.9061**	3.5483	**120.54**	3578.53	**4.6710**	2.1783
100 x 5	**-0.0457***	0.0641	**60.98**	4257.50	**0.5592**	0.4498
100 x 10	**0.2727**	1.3547	**122.16**	5126.50	**2.1012**	1.0470
100 x 20	**0.9476**	3.2464	**244.90**	6517.82	**3.4743**	1.2763
200 x 10	**0.2673**	0.9406	**253.06**	10244.68	**0.9901**	0.3257
200 x 20	**1.1837**	2.8140	**521.19**	13683.71	**2.2499**	0.6756
500 x 20	**0.9202**	1.5812	**1825.29**	35764.19	**0.7512**	0.1014
média	**0.3905**	1.4360	**276.88**	7258.82	**2.6036**	1.6029

No DRP tem-se que o IG_{im} alcançou um resultado melhor do que *Upper Bound* em 3 oportunidades como pode ser visto nos valores marcados com um asterisco indicados na Tabela 23. Na Figura 8 é ilustrado que o algoritmo BMH obteve um resultado próximo, mas nunca melhor, ao IG_{BLSP} apenas em 3 conjuntos de instâncias (20x5, 50x5 e 100x5). Já nos conjuntos 50x10, 50x20, 100x20 e 200x20 a diferença entre os resultados obtidos pelos dois métodos é muito grande em favor do IG_{BLSP}.

A capacidade de um algoritmo melhorar a solução inicial também é algo de suma importância. Por isso foi realizada uma comparação entre os algoritmos para verificar quanto cada um melhorou a solução inicial NEH. A Tabela 23 e a Figura 9 mostram que em todos os conjuntos de instância, o IG_{BLSP} foi superior ao BMH, sendo que nos conjuntos 50x10, 50x20, 100x10, 100x20 e 200x20 a diferença entre os dois métodos ficou mais nítida em prol do IG_{BLSP}. Na média, o IG_{im} conseguiu melhorar a solução do NEH em 2,60% contra 1,60% do BMH_{im}, dando uma diferença final entre os algoritmos de aproximadamente 1%.

Figura 8 – Comparação do Desvio Relativo Percentual para cada conjunto de instâncias entre os algoritmos IG_{im} e BMH_{im}.

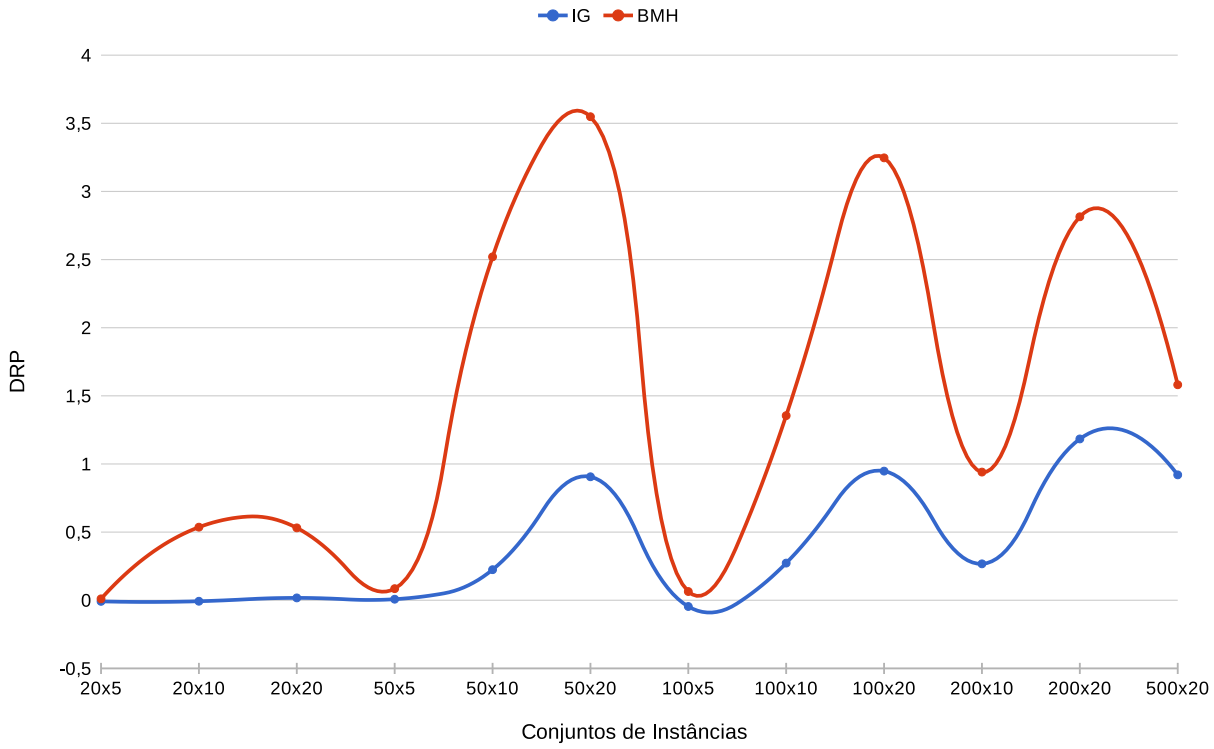
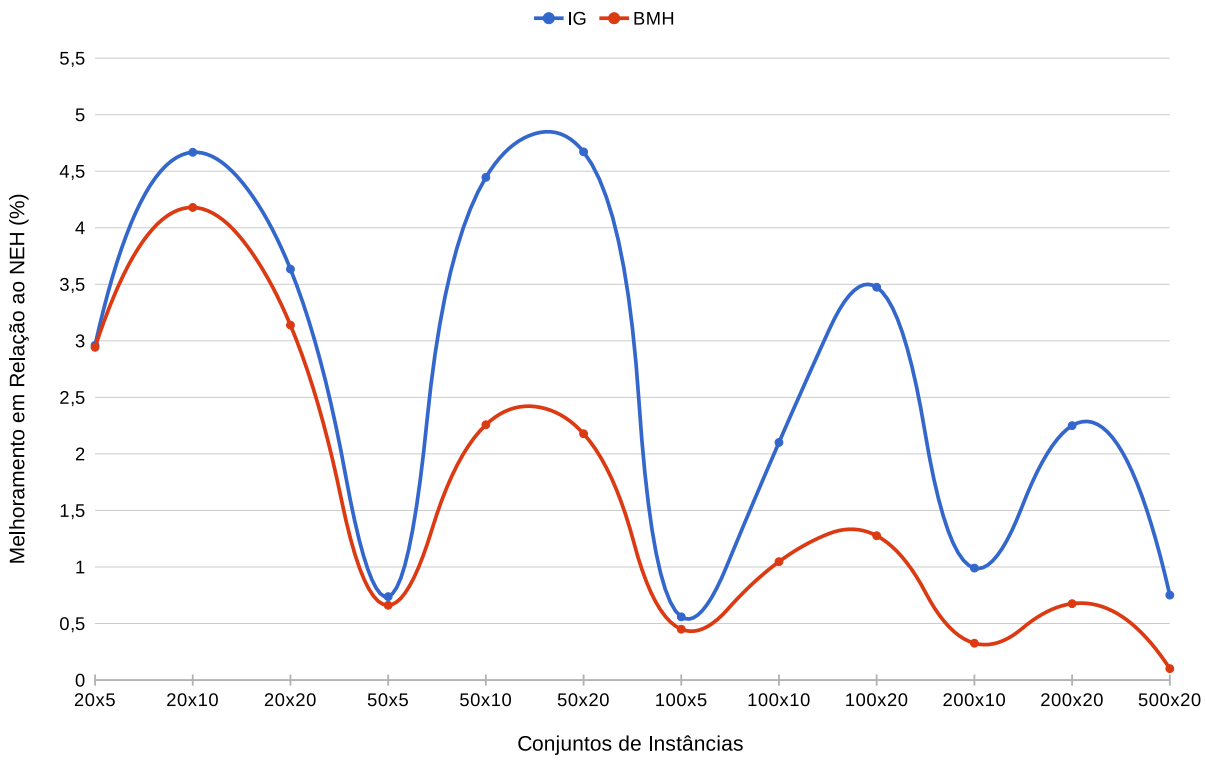
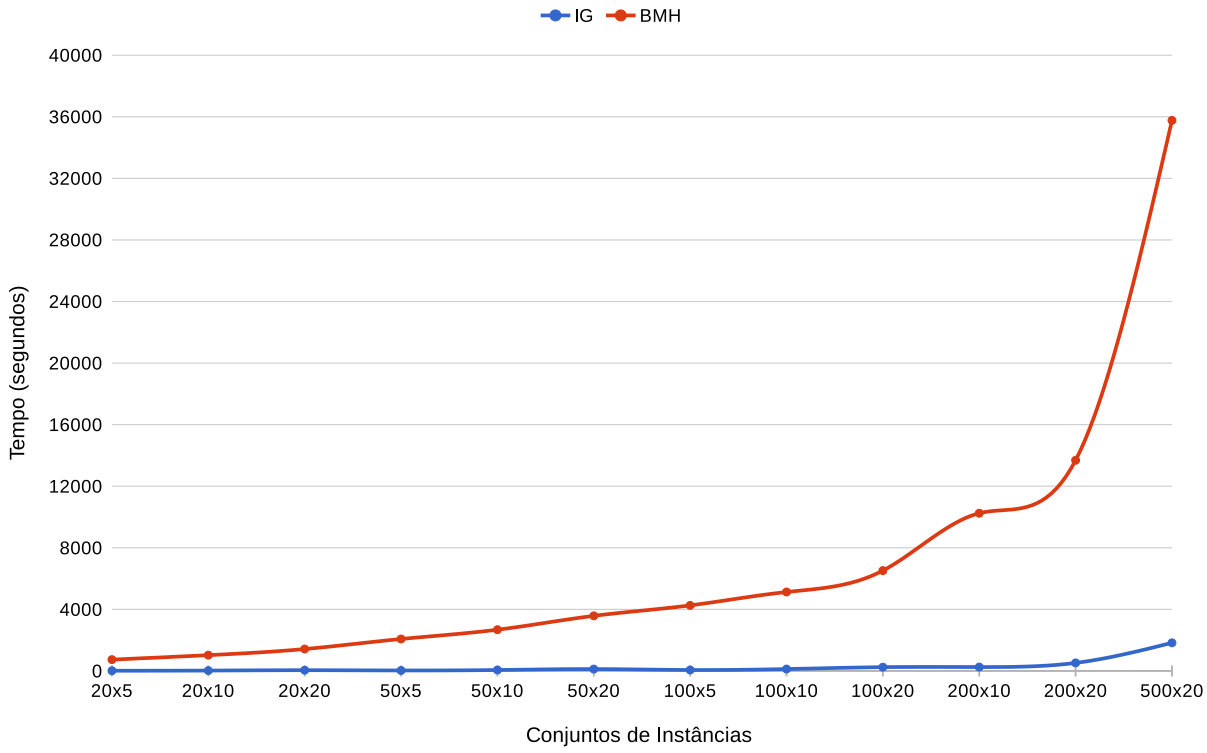


Figura 9 – Comparação do melhoramento em relação a heurística inicial NEH obtido pelos algoritmos IG_{im} e BMH_{im} em cada conjunto de instâncias.



Um dos pontos chaves de um algoritmo de otimização combinatória juntamente com a qualidade da solução é o tempo de execução. A Tabela 23 e a Figura 10 nos mostram os dados referentes ao tempo de execução dos dois algoritmos aqui implementados. Enquanto o IG_{im} teve uma curva quase linear, o algoritmo BMH_{im} apresentou uma curva próxima a exponencial. A diferença do tempo de execução entre os dois métodos é muito grande, um exemplo é que nem mesmo no conjunto de instâncias 500x20, tido como o mais complexo, o IG_{im} demorou em média mais do que 3600 segundos (1 hora) para executar uma instância, já o algoritmo BMH_{im} no conjunto 50x20 demorou em média 1 hora aproximadamente e no 500x20 o tempo médio foi de aproximadamente 10 horas para executar uma instância.

Figura 10 – Comparação da média dos tempos de execução em cada conjunto de instâncias para os algoritmos IG_{im} e BMH_{im}.



Diante das tabelas e figuras apresentadas nesta seção tem-se que a heurística IG_{BLSP} proposta por Dubois-Lacoste, Pagnozzi e Stützle (2017) apresenta um custo-benefício amplamente superior em relação a meta-heurística BMH desenvolvida por Marichelvam, Tosun e Geetha (2017), pois com relação à qualidade da solução, o algoritmo IG_{BLSP} se mostrou superior ao BMH. Já com relação ao tempo de execução, o IG_{BLSP} conseguiu concluir a execução das instâncias em um tempo muito inferior ao BMH que por sua vez se mostrou ineficiente neste quesito.

6 Conclusão

Este trabalho abordou o Problema do Flow Shop Permutacional (PFSP) com a função objetivo de minimizar o *makespan*.

Através de uma Revisão Sistemática da Literatura (RSL) realizada no Capitulo 3, alguns trabalhos de outros autores foram selecionados para uma leitura profunda com a finalidade de se escolher dois algoritmos que se propõem a resolver o problema citado acima. O intuito deste trabalho foi saber qual dos dois métodos apresenta um melhor custo-benefício ao resolver tal problema.

Antes de escolher os dois algoritmos para a comparação, a RSL aqui realizada chegou a duas conclusões importantes. A primeira foi que nenhum trabalho escolhido para a leitura intermediária utilizou o processo de revisão sistemática durante o processo de pesquisa. A segunda foi que apenas uma dentre as 38 publicações escolhidas para a leitura intermediária disponibilizou o código fonte de seus algoritmos. A falta destes dados prejudica a repetibilidade e consequentemente o avanço científico, como mencionado no Capitulo 1, e esta é uma falha grave na área da computação.

Por fim, através dos dados coletados na RSL foram selecionadas a heurística Iteração Gulosa com Busca Local em Soluções Parciais (IG_{BLSP}) e a meta-heurística Busca do Macaco Híbrida (BMH) para as implementações que podem ser encontradas no Capitulo 4.

O Objetivo Específico 1 foi alcançado por meio da subseção 5.3.1, onde por meio de um estudo comparativo entre os dois métodos foi concluído que o algoritmo implementado neste trabalho obteve valores DRP muito próximos ao algoritmo original.

O Objetivo Específico 2 foi atingido na subseção 5.3.2, onde foi realizada uma comparação entre os dois métodos e a conclusão foi que o algoritmo implementado neste trabalho obteve resultados do DRP próximos ao algoritmo original.

O Objetivo Específico 3 também foi completado a partir da subseção 5.3.3, onde os dois métodos implementados nesta monografia foram confrontados entre si. A conclusão foi que o algoritmo implementado IG_{BLSP} apresenta um custo-benefício amplamente superior ao método BMH, principalmente em relação ao Tempo de Execução (TE), onde para a instância mais simples (20x5) proposta por Taillard a diferença da média do TE entre os dois algoritmos foi de aproximadamente 721 segundos (12 minutos) em prol do IG_{BLSP}. Já para a instância mais complexa (500x20) a diferença da média do TE entre os dois algoritmos foi de 33939 segundos (9,42 horas) em prol do IG_{BLSP}. Com relação a qualidade da solução o IG_{BLSP} também foi melhor que o BMH em todas as instâncias analisadas.

Diante da resolução dos objetivos específicos tem-se que a hipótese registrada na seção 1.2 se mostrou verdadeira. De fato com relação a qualidade da solução os métodos se mostraram eficazes e encontraram soluções próximas do ótimo, com ligeira vantagem para o IG_{BLSP}. Já com relação ao tempo de execução a discrepância entre os algoritmos foi clara, com grande vantagem para o IG_{BLSP}, isto é, o tempo de processamento consumido por este foi menor que o BMH, confirmando assim a hipótese apresentada neste trabalho.

6.1 Contribuições

As principais contribuições deste trabalho foram para a área acadêmica ao se realizar uma revisão sistemática da literatura sobre o PFSP e também para os pesquisadores e industrias que trabalham com o PFSP ao se analisar e demonstrar que o algoritmo IG_{BLSP} possui um custo-benefício muito superior ao BMH.

6.2 Trabalhos Futuros

Finalmente tem-se como proposta para trabalhos futuros:

- Alterar alguns parâmetros e fluxos do algoritmo BMH para o tornar mais simples e com um maior potencial de alcançar melhores qualidades de soluções para o PFSP.
- Utilizar técnicas de paralelização na meta-heurística BMH com a finalidade de reduzir o tempo de execução do método.
- Utilizar técnicas de paralelização na heurística IG_{BLSP} objetivando conseguir executar um maior número de iterações e consequentemente melhorar a qualidade da solução.
- Utilizar o *benchmark* proposto por Vallada, Ruiz e Framinan (2015), pois segundo o autor, este conjunto de instâncias pode ser usado para estabelecer diferenças estatísticas entre os algoritmos de uma forma sólida pelo fato de ser mais difícil e possuir um maior poder discriminativo quando comparado com outros *benchmarks*.

Referências

AURICH, P. et al. Simulation-based optimization for solving a hybrid flow shop scheduling problem. In: *Proceedings of the 2016 Winter Simulation Conference*. Piscataway, NJ, USA: IEEE Press, 2016. (WSC '16), p. 2809–2819. Disponível em: <http://dl.acm.org/citation.cfm?id=3042409.3042444>. Citado 3 vezes nas páginas 37, 40 e 43.

BALAJI, A.; PORSELVI, S. Artificial immune system algorithm and simulated annealing algorithm for scheduling batches of parts based on job availability model in a multi-cell flexible manufacturing system. *Procedia Engineering*, v. 97, p. 1524 – 1533, 2014. ISSN 1877-7058. 12th Global Congress on Manufacturing and Management {GCMM} - 2014. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877705814035048>. Citado 3 vezes nas páginas 37, 38 e 42.

BEASLEY, J. E. Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, Springer, v. 41, n. 11, p. 1069–1072, 1990. Citado na página 42.

BOŻEJKO, W. et al. Cyclic hybrid flow-shop scheduling problem with machine setups. *Procedia Computer Science*, v. 29, p. 2127 – 2136, 2014. ISSN 1877-0509. 2014 International Conference on Computational Science. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050914003743>. Citado 3 vezes nas páginas 37, 38 e 42.

CAMPOS, S. C. Aplicação de metaheurísticas para o problema de programação da produção em ambiente assembly flowshop com três estágios e tempos de preparação dependentes da sequência. Universidade Federal de Viçosa, 2015. Disponível em: <http://www.locus.ufv.br/handle/123456789/6666>. Citado 3 vezes nas páginas 37, 39 e 42.

CAMPOS, S. C.; ARROYO, J. E. C. Nsga-ii with iterated greedy for a bi-objective three-stage assembly flowshop scheduling problem. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2014. (GECCO '14), p. 429–436. ISBN 978-1-4503-2662-9. Disponível em: <http://doi.acm.org/10.1145/2576768.2598324>. Citado 3 vezes nas páginas 37, 38 e 42.

CARLIER, J. Ordonnancements a contraintes disjonctives. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, v. 12, n. 4, p. 333–350, 1978. Citado 2 vezes nas páginas 42 e 43.

CARNIEL, G. C. Including workers with disabilities in flow shop scheduling. Universidade Federal do Rio Grande do Sul, 2015. Disponível em: <http://hdl.handle.net/10183/132054>. Citado 3 vezes nas páginas 37, 39 e 42.

CHAKROUN, I.; MELAB, N. Operator-level gpu-accelerated branch and bound algorithms. *Procedia Computer Science*, v. 18, p. 280 – 289, 2013. ISSN 1877-0509. 2013 International Conference on Computational Science. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050913003347>. Citado 3 vezes nas páginas 37, 38 e 42.

COLLBERG, C.; PROEBSTING, T. A. Repeatability in computer systems research. *Communications of the ACM*, ACM, v. 59, n. 3, p. 62–69, 2016. Citado 2 vezes nas páginas 15 e 16.

DAS, S. R.; CANEL, C. An algorithm for scheduling batches of parts in a multi-cell flexible manufacturing system. *International Journal of Production Economics*, Elsevier, v. 97, n. 3, p. 247–262, 2005. Citado na página 42.

DEMIRKOL, E.; MEHTA, S.; UZSOY, R. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, Elsevier, v. 109, n. 1, p. 137–141, 1998. Citado na página 43.

DUBOIS-LACOSTE, J.; PAGNOZZI, F.; STÜTZLE, T. An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. *Computers & Operations Research*, Elsevier, v. 81, p. 160–166, 2017. Citado 10 vezes nas páginas 14, 37, 43, 44, 49, 53, 81, 82, 83 e 99.

EDDALY, M.; JARBOUI, B.; SIARRY, P. Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem. *Journal of Computational Design and Engineering*, v. 3, n. 4, p. 295 – 311, 2016. ISSN 2288-4300. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2288430016300161>. Citado 3 vezes nas páginas 37, 41 e 43.

FERNANDEZ-VIAGAS, V.; RUIZ, R.; FRAMINAN, J. M. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research*, Elsevier, v. 257, n. 3, p. 707–721, 2016. Citado 2 vezes nas páginas 29 e 30.

FERONE, D. et al. Combining simulation with a grasp metaheuristic for solving the permutation flow-shop problem with stochastic processing times. In: *Proceedings of the 2016 Winter Simulation Conference*. Piscataway, NJ, USA: IEEE Press, 2016. (WSC '16), p. 2205–2215. ISBN 978-1-5090-4484-9. Disponível em: <http://dl.acm.org/citation.cfm?id=3042094.3042369>. Citado 3 vezes nas páginas 37, 41 e 43.

GAREY, M. R.; JOHNSON, D. S.; SETHI, R. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, INFORMS, v. 1, n. 2, p. 117–129, 1976. Citado na página 30.

HELLER, J. Some numerical experiments for an m× j flow shop and its decision-theoretical aspects. *Operations Research*, INFORMS, v. 8, n. 2, p. 178–184, 1960. Citado na página 43.

HOLANDA, T. C. O relacionamento do problema de sequenciamento clássico com o problema do caixeiro viajante e sua resolução numa abordagem evolutiva. Universidade Federal do Ceará, 2015. Disponível em: <http://www.teses.ufc.br/tde_busca/arquivo.php?codArquivo=15685>. Citado 3 vezes nas páginas 37, 39 e 42.

HORDONES, P. A.; CAMARGO, V. H.; FUCHIGAMI, H. Y. Programação da produção em flow shop permutacional envolvendo medidas de atraso: Uma constribuição bibliométrica. 2016. Citado 2 vezes nas páginas 17 e 27.

IBRAHEM, A.; ELMEKKAWY, T.; PENG, Q. Robust metaheuristics for scheduling cellular flowshop with family sequence-dependent setup times. *Procedia {CIRP}*, v. 17, p. 428 – 433, 2014. ISSN 2212-8271. Variety Management in ManufacturingProceedings of the 47th {CIRP} Conference on Manufacturing Systems. Disponível em: <http:

//www.sciencedirect.com/science/article/pii/S2212827114003229>. Citado 3 vezes nas páginas 37, 39 e 42.

JAROSŁAW, P.; CZESŁAW, S.; DOMINIK, Ż. Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm. *Procedia Computer Science*, v. 18, p. 936 – 945, 2013. ISSN 1877-0509. 2013 International Conference on Computational Science. Disponível em: <http://www.sciencedirect.com/science/article/pii/S187705091300402X>. Citado 3 vezes nas páginas 37, 38 e 42.

JOHNSON, S. M. Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics (NRL)*, Wiley Online Library, v. 1, n. 1, p. 61–68, 1954. Citado na página 17.

JOLAI, F. et al. Bi-objective simulated annealing approaches for no-wait two-stage flexible flow shop scheduling problem. *Scientia Iranica*, v. 20, n. 3, p. 861 – 872, 2013. ISSN 1026-3098. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1026309813000461>. Citado 3 vezes nas páginas 37, 38 e 42.

KELLER, F.; SCHÖNBORN, C.; REINHART, G. Energy-orientated machine scheduling for hybrid flow shops. *Procedia {CIRP}*, v. 29, p. 156 – 161, 2015. ISSN 2212-8271. The 22nd {CIRP} Conference on Life Cycle Engineering. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2212827115001791>. Citado 3 vezes nas páginas 37, 39 e 42.

KIESKOSKI, A. Um estudo do problema de flow shop permutacional, uma proposta de solução através da metaheurística colônia de formigas. 2016. Disponível em: <http://dspace.c3sl.ufpr.br:8080/dspace/handle/1884/45527>. Citado 3 vezes nas páginas 37, 41 e 43.

KOMESU, A. S. Heurística evolutiva para a minimização do atraso total em ambiente de produção flow shop com buffer zero. Universidade de São Paulo, 2015. Disponível em: <http://www.teses.usp.br/teses/disponiveis/18/18156/tde-15062015-100735/>. Citado 3 vezes nas páginas 37, 40 e 42.

LEEFTINK, A. G. et al. Batch scheduling in the histopathology laboratory. *Flexible Services and Manufacturing Journal*, p. 1–27, 2016. ISSN 1936-6590. Disponível em: <http://dx.doi.org/10.1007/s10696-016-9257-3>. Citado 3 vezes nas páginas 37, 41 e 43.

LEEUWEN, J. van. Algorithms and complexity. Elsevier, 1998. Citado na página 13.

LI, J.; PAN, Q. k.; DUAN, P. y. An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping. *IEEE Transactions on Cybernetics*, v. 46, n. 6, p. 1311–1324, June 2016. ISSN 2168-2267. Citado 3 vezes nas páginas 37, 41 e 43.

LI, W.; DAI, H.; ZHANG, D. The relationship between maximum completion time and total completion time in flowshop production. *Procedia Manufacturing*, v. 1, p. 146 – 156, 2015. ISSN 2351-9789. 43rd North American Manufacturing Research Conference, {NAMRC} 43, 8-12 June 2015, {UNC} Charlotte, North Carolina, United States. Disponível em: <http://www.sciencedirect.com/science/article/pii/S235197891501077X>. Citado 3 vezes nas páginas 37, 40 e 43.

LI, X. et al. Trajectory scheduling methods for minimizing total tardiness in a flowshop. *Operations Research Perspectives*, v. 2, p. 13 – 23, 2015. ISSN 2214-7160. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2214716015000020>. Citado 3 vezes nas páginas 37, 40 e 43.

LI, X.; MA, S. Multi-objective memetic search algorithm for multi-objective permutation flow shop scheduling problem. *IEEE Access*, v. 4, p. 2154–2165, 2016. ISSN 2169-3536. Nawaz–Enscore–Hoam é utilizado para inicializar a população. Citado 3 vezes nas páginas 37, 41 e 43.

LUGO, P. L. M. Programação da produção em sistemas flowshop híbrido com buffers limitados. Universidade Federal de São Carlos, 2013. Disponível em: <http://www.bdtd.ufscar.br/htdocs/tedeSimplificado//tde_busca/arquivo.php?codArquivo=7731>. Citado 3 vezes nas páginas 37, 38 e 42.

MANSOURI, S. A.; AKTAS, E.; BESIKCI, U. Green scheduling of a two-machine flowshop: Trade-off between makespan and energy consumption. *European Journal of Operational Research*, v. 248, n. 3, p. 772 – 788, 2016. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221715008206>. Citado 3 vezes nas páginas 37, 41 e 43.

MARICHELVAM, M.; TOSUN, Ö.; GEETHA, M. Hybrid monkey search algorithm for flow shop scheduling problem under makespan and total flow time. *Applied Soft Computing*, Elsevier, v. 55, p. 82–92, 2017. Citado 11 vezes nas páginas 7, 14, 37, 43, 44, 57, 81, 84, 85, 87 e 99.

MARTIN, S. et al. A multi-agent based cooperative approach to scheduling and routing. *European Journal of Operational Research*, v. 254, n. 1, p. 169 – 178, 2016. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221716300984>. Citado 4 vezes nas páginas 37, 41, 43 e 44.

MINELLA, G.; RUIZ, R.; CIAVOTTA, M. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, INFORMS, v. 20, n. 3, p. 451–471, 2008. Citado na página 42.

MIRABI, M.; GHOMI, S. M. T. F.; JOLAI, F. A novel hybrid genetic algorithm to solve the make-to-order sequence-dependent flow-shop scheduling problem. *Journal of Industrial Engineering International*, v. 10, n. 2, p. 57, 2014. ISSN 2251-712X. Disponível em: <http://dx.doi.org/10.1007/s40092-014-0057-7>. Citado 3 vezes nas páginas 37, 39 e 42.

MOHAMMADI, M. et al. Rolling-horizon and fix-and-relax heuristics for the multi-product multi-level capacitated lotsizing problem with sequence-dependent setups. *Journal of Intelligent Manufacturing*, Springer, v. 21, n. 4, p. 501–510, 2010. Citado na página 42.

MORITZ, R. L. et al. Refined ranking relations for multi objective optimization andapplication to p-aco. In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2013. (GECCO '13), p. 65–72. ISBN 978-1-4503-1963-8. Disponível em: <http://doi.acm.org/10.1145/2463372.2463388>. Citado 3 vezes nas páginas 37, 38 e 42.

MORITZ, R. L. et al. Refined ranking relations for selection of solutions in multi objective metaheuristics. *European Journal of Operational Research*, v. 243, n. 2, p. 454 – 464, 2015.

ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221714008662>. Citado 3 vezes nas páginas 37, 40 e 43.

MULROW, C. D. Rationale for systematic reviews. *BMJ: British Medical Journal*, BMJ Group, v. 309, n. 6954, p. 597, 1994. Citado na página 31.

NADERI, B.; ZANDIEH, M.; ROSHANAEI, V. Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness. *The International Journal of Advanced Manufacturing Technology*, v. 41, n. 11, p. 1186–1198, 2009. ISSN 1433-3015. Disponível em: <http://dx.doi.org/10.1007/s00170-008-1569-3>. Citado na página 43.

NAWAZ, M.; ENSCORE, E. E.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, Elsevier, v. 11, n. 1, p. 91–95, 1983. Citado 4 vezes nas páginas 29, 45, 50 e 58.

NOURI, N.; LADHARI, T. Minimizing regular objectives for blocking permutation flow shop scheduling: Heuristic approaches. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2015. (GECCO '15), p. 441–448. ISBN 978-1-4503-3472-3. Disponível em: <http://doi.acm.org/10.1145/2739480.2754638>. Citado 3 vezes nas páginas 37, 40 e 43.

PINEDO, M. Scheduling: Theory, algorithms, and systems. Springer New York, 2012. Disponível em: <https://books.google.com.br/books?id=QRiDnuXSnVwC>. Acesso em: 13/11/2017. Citado 6 vezes nas páginas 17, 22, 23, 24, 25 e 26.

PUGAZHENTHI, R.; XAVIOR, M. A. A genetic algorithm applied heuristic to minimize the makespan in a flow shop. *Procedia Engineering*, v. 97, p. 1735 – 1744, 2014. ISSN 1877-7058. 12th Global Congress on Manufacturing and Management (GCMM) - 2014. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877705814033931>. Citado 3 vezes nas páginas 37, 39 e 42.

REEVES, C. R. A genetic algorithm for flowshop sequencing. *Computers & operations research*, Elsevier, v. 22, n. 1, p. 5–13, 1995. Citado na página 43.

ROJAS, A. J. B. Heuristics for flow shop scheduling : considering non-permutation schedules and a heterogeneous workforce. Universidade Federal do Rio Grande do Sul, 2015. Disponível em: <http://hdl.handle.net/10183/131868>. Citado 3 vezes nas páginas 37, 40 e 43.

RUIZ, R.; MAROTO, C. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, Elsevier, v. 165, n. 2, p. 479–494, 2005. Citado na página 28.

RUIZ, R.; MAROTO, C.; ALCARAZ, J. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, v. 165, n. 1, p. 34 – 54, 2005. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221704000529>. Citado na página 43.

RUIZ, R.; ŞERIFOĞLU, F. S.; URLINGS, T. Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, Elsevier, v. 35, n. 4, p. 1151–1175, 2008. Citado na página 42.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, Elsevier, v. 177, n. 3, p. 2033–2049, 2007. Citado 3 vezes nas páginas 29, 44 e 49.

RUIZ, R.; STüTZLE, T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, v. 187, n. 3, p. 1143 – 1159, 2008. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221706008277>. Citado na página 43.

SADAQA, M.; MORAGA, R. J. Scheduling blocking flow shops using meta-raps. *Procedia Computer Science*, v. 61, p. 533 – 538, 2015. ISSN 1877-0509. Complex Adaptive Systems San Jose, {CA} November 2-4, 2015. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050915030410>. Citado 3 vezes nas páginas 37, 40 e 43.

SIMÕES, W. L. Abordagem metaheurística híbrida para a otimização de sequenciamento de produção em flow shop permutacional com tempos de setup dependentes da sequência. Universidade do Vale do Rio dos Sinos, 2016. Disponível em: <http://www.repositorio.jesuita.org.br/handle/UNISINOS/6022>. Citado 3 vezes nas páginas 37, 41 e 43.

SIOUD, A.; GAGNÉ, C.; GRAVEL, M. An ant colony optimization for solving a hybrid flexible flowshop. In: *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2014. (GECCO Comp '14), p. 17–18. ISBN 978-1-4503-2881-4. Disponível em: <http://doi.acm.org/10.1145/2598394.2598402>. Citado 3 vezes nas páginas 37, 39 e 42.

STÜTZLE, T. Applying iterated local search to the permutation flow shop problem. *FG Intellektik, TU Darmstadt, Darmstadt, Germany*, 1998. Citado na página 29.

TAILLARD, E. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, v. 47, n. 1, p. 65 – 74, 1990. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/037722179090090X>. Citado na página 43.

TAILLARD, E. Benchmarks for basic scheduling problems. *european journal of operational research*, Elsevier, v. 64, n. 2, p. 278–285, 1993. Citado 7 vezes nas páginas 28, 39, 42, 43, 44, 81 e 82.

TASGETIREN, M. F. et al. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European journal of operational research*, Elsevier, v. 177, n. 3, p. 1930–1947, 2007. Citado na página 63.

VALLADA, E.; RUIZ, R.; FRAMINAN, J. M. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, Elsevier, v. 240, n. 3, p. 666–677, 2015. Citado 2 vezes nas páginas 30 e 101.

VILLADIEGO, H. M. M. Heurísticas para o problema de dimensionamento e sequenciamento de lotes em um ambiente de produção flowshop. Universidade Federal de Viçosa, 2014. Disponível em: <http://locus.ufv.br/handle/123456789/2682>. Citado 3 vezes nas páginas 37, 39 e 42.

YAHYAOUI, H. et al. A hybrid ils-vnd based hyper-heuristic for permutation flowshop scheduling problem. *Procedia Computer Science*, v. 60, p. 632 – 641, 2015. ISSN 1877-0509. Knowledge-Based and Intelligent Information & Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050915023261>. Citado 3 vezes nas páginas 37, 40 e 43.

YANG, Z.; MA, Z.; WU, S. Optimized flowshop scheduling of multiple production lines for precast production. *Automation in Construction*, v. 72, Part 3, p. 321 – 329, 2016. ISSN 0926-5805. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0926580516301753>. Citado 3 vezes nas páginas 37, 42 e 43.

YENISEY, M. M.; YAGMAHAN, B. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega*, v. 45, p. 119 – 135, 2014. ISSN 0305-0483. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305048313000832>. Citado 2 vezes nas páginas 26 e 29.

ZHAO, R.; TANG, W. Monkey algorithm for global numerical optimization. *Journal of Uncertain Systems*, v. 2, n. 3, p. 165–176, 2008. Citado na página 57.

Apêndices

APÊNDICE A – Códigos da configuração do projeto, do *core* e do parser utilizados nos algoritmos

A.1 Arquivo project.clj

```
(defproject BMH-IG "PFSP"
  :description "Descrição: Algoritmos BMH e IG_BLSP"
  :url "https://github.com/ViniciusTxr/algoritmos-flowshop"
  :license {:name "Public License"}
  :dependencies [[org.clojure/clojure "1.8.0"]]
  :main ^:skip-aot core
  :repl-options {:timeout 2000000}
  :target-path "target/%s"
  :jvm-opts ["-Xmx4g" "-server"]
  :profiles {:uberjar {:aot :all}})
```

A.2 Arquivo core.clj

```
(ns core
  (:require [NEH :as neh]
            [monkey-search :as ms]
            [iterated-greedy :as ig]
            [parser :as p])
  (:gen-class))

(def form-data (java.text.SimpleDateFormat. "dd-MM-yyyy HH:mm:ss"))
(def chamadas 10)

(defn monkey
  "Função que calcula o resultado para o algoritmo Busca do Macaco Híbrida (Monkey
   ↪  Search) e salva a resposta em arquivo"
  [i melhor pior media-make media-tempo]
    (let [
          data-inicio (java.util.Date.)
          tempo-inicio (System/currentTimeMillis)
          inicial (ms/solucao_inicial)
          resultado (time (ms/monkey-search 0 inicial (ms/encontra-melhor inicial)))
          tempo-fim (System/currentTimeMillis)
```

```
          data-fim (java.util.Date.)
          tempo-total (float(/ (- tempo-fim tempo-inicio) 1000))
          make-result (first resultado)
          melhor (if (< make-result melhor) make-result melhor)
          pior (if (> make-result pior) make-result pior)
          media-make (+ media-make make-result)
          media-tempo (+ tempo-total media-tempo)
          ]
      (with-open [w (clojure.java.io/writer "src/resultados_BMH.txt" :append true)]
        (.write w (str "Makespan: "make-result "\nPermutação: "(into [] (map
        ↪  #(Integer/parseInt %) (map #(subs % 1) (map name (map first (first
        ↪  (ms/smallest-position-value (vector (second resultado)))))))))"\nData
        ↪  Inicio: " (.format form-data data-inicio) " / Data Fim: " (.format
        ↪  form-data data-fim) "\nTempo: " tempo-total " segundos\n\n")))
      (if(= i chamadas)
        (with-open [w (clojure.java.io/writer "src/resultados_BMH.txt" :append
        ↪  true)]
          (.write w (str "Melhor: "melhor "\nPior: "pior "\nMedia makespan: "(/
          ↪  media-make chamadas) "\nMedia tempo: "(/ (double (long (* (/
          ↪  media-tempo chamadas) 100))) 100) " segundos\n\n\n"))))
      (if (< i chamadas)
          (recur (inc i) melhor pior media-make media-tempo))))

(defn iterated
  "Função que calcula o resultado para o algoritmo Iteração Gulosa (Iterated Greedy)
  ↪  e salva a resposta em arquivo"
  [i melhor pior media-make media-tempo media-iter]
    (let [
          data-inicio (java.util.Date.)
          tempo-inicio (System/currentTimeMillis)
          resultado (time (ig/iterated-greedy 0 neh/NEH neh/NEH (+ ig/tempo
          ↪  (System/currentTimeMillis))))
          tempo-fim (System/currentTimeMillis)
          data-fim (java.util.Date.)
          tempo-total (float(/ (- tempo-fim tempo-inicio) 1000))
          make-result (first (first resultado))
          melhor (if (< make-result melhor) make-result melhor)
          pior (if (> make-result pior) make-result pior)
          media-make (+ media-make make-result)
          media-tempo (+ tempo-total media-tempo)
          iteracao (+ 1 (second resultado))
          media-iter (+ iteracao media-iter)
          ]
      (with-open [w (clojure.java.io/writer "src/resultados_IG.txt" :append true)]
```

```
          (.write w (str "Makespan: "make-result "\nPermutação: "(into [] (map
           ↪  #(Integer/parseInt %) (map #(subs % 1) (map name (second (first
           ↪  resultado)))))) "\nIterações: " iteracao "\nData Inicio: " (.format
           ↪  form-data data-inicio) " / Data Fim: " (.format form-data data-fim)
           ↪  "\nTempo: " tempo-total " segundos\n\n")))
      (if (= i chamadas)
        (with-open [w (clojure.java.io/writer "src/resultados_IG.txt" :append true)]
          (.write w (str "Melhor: "melhor "\nPior: "pior "\nMedia makespan: "(/
           ↪  media-make chamadas)"\nMedia de iteraçoes: "(/ media-iter chamadas)
           ↪  "\nMedia tempo: "(/ (double (long (* (/ media-tempo chamadas) 100)))
           ↪  100) " segundos\n\n\n"))))
      (if (< i chamadas)
        (recur (inc i) melhor pior media-make media-tempo media-iter))))

(defn -main
  "opcao passada como parametro, se 1 entao chama o Busca do Macaco Híbrida (Monkey
   ↪  Search), se 2 entao chama o Iteração Gulosa (Iterated Greedy), se outra entao
   ↪  não faz nada."
  [opc]
    (let [
          opc (Integer/parseInt opc)
          resultado (case opc
                      1 "Busca do Macaco Híbrida - Monkey Search"
                      2 "Iteração Gulosa - Iterated Greedy"
                      "Opção Invalida!")
          ]
      (if(or (= opc 1) (= opc 2))
        (do
          (println "Algoritmo: "resultado " | Instancia: "p/instancia)
           (if (= opc 2)
            (do
              (with-open [w (clojure.java.io/writer "src/resultados_IG.txt" :append
               ↪  true)]
                (.write w (str "---------------------------------------\n" "Numero de
                 ↪  trabalhos: " p/n_job "\nNumero de maquinas: " p/n_maq
                 ↪  "\nInstancia: "p/instancia"\nLower bound: "p/lower_bound
                 ↪  "\nUpper bound: "p/upper_bound "\n\n")))
              (iterated 1 999999999 0 0.0 0.0 0.0))
            (do
              (with-open [w (clojure.java.io/writer "src/resultados_BMH.txt" :append
               ↪  true)]
                (.write w (str "---------------------------------------\n" "Numero de
                 ↪  trabalhos: " p/n_job "\nNumero de maquinas: " p/n_maq
                 ↪  "\nInstancia: "p/instancia"\nLower bound: "p/lower_bound "\nUpper
                 ↪  bound: "p/upper_bound "\n\n")))
              (monkey 1 999999999 0 0.0 0.0)))
        )resultado)))
```

A.3 Arquivo parser.clj

```
(ns parser
  (:require [clojure.string :as s]))

(defn le_arquivo
  "le o arquivo das instancias e os salva em uma string, salva a string em uma
   ↪  lista, onde os elementos estao separados de acordo com as linhas do aqruivo "
  [ ]
  (let [string1 (slurp "src/Benchmarks/tai20_5.txt")]
  (s/split-lines string1)))

(defn strInt
  "converte string para inteiro"
  [str]
  (Integer/parseInt str))

(def instancia 1)

(def n_maq (strInt ((let [inicio ((le_arquivo) 1)
                                aux (s/split inicio #"\s+")
                                ] aux) 2)))

(def linha_inicial (+ 1 (* (- instancia 1) (+ n_maq 3))))

(defn separa-arquivos
  "separa e coloca cada parte do arquivo de entrada em uma variável específica"
  [ ]
  (let [inicio ((le_arquivo) linha_inicial)
        aux (s/split inicio #"\s+")]
    (hash-map :n_maq (strInt (aux 2))
              :n_trab (strInt (aux 1))
              :seed (strInt (aux 3))
              :ub (strInt (aux 4))
              :lb (strInt (aux 5)))))

(def n_job ((find (separa-arquivos) :n_trab) 1))
(def upper_bound ((find (separa-arquivos) :ub) 1))
(def lower_bound ((find (separa-arquivos) :lb) 1))
(def prim_linha (+ (* n_maq  (- instancia 1)) (* 3 (+ (- instancia 1) 1))))
(def ultim_linha (+ prim_linha n_maq))

(defn pega-matriz
  "pega a matriz do arquivo, removendo os espaços nos caracteres"
  [ ]
  (map #(subvec % 1 (+ n_job 1))
       (map #(s/split % #"\s+")
```

```
            (subvec (le_arquivo) prim_linha ultim_linha))))

(defn transforma
  "transforma os elementos char pra inteiro"
 [matriz]
 (into [ ] (map #(Integer/parseInt %) matriz)))

(defn gera-lista
  "gera a lista final dos tempos de trabalho "
  [ ]
  (into [ ] (map transforma (pega-matriz))))

(def ger-list (gera-lista))

(defn transposicao-matriz
  "pega a matriz e faz a transposição"
  [ ]
  (into [ ] (apply map vector ger-list)))

(defn key-jobs
  "constroi a lista de keywords de acordo com o numero de trabalhos"
  [ ]
  (into [ ] (map keyword (map #(str "j"%) (range 1 (+ n_job 1))))))

(defn mapeia-trabalhos
  "concatena as keywords com a matriz de tempo de processamento do trabalho"
  [ ]
  (zipmap (key-jobs) (transposicao-matriz)))

(defn remove-elem-vetor
  "remove elem in coll"
  [vetor pos]
  (vec (concat (subvec vetor 0 pos) (subvec vetor (inc pos)))))

(defn add-vec
  "add um elemento (elem) a um vetor (vet) na posiçao (pos)"
  [vet elem pos]
  (let [
        tam-vet (count vet)
        novo_vet (if-let [_ (= pos 0)]
                   (into [ ] (cons elem vet))
                   (if-let [_ (= tam-vet pos)]
                       (conj vet elem)
                       (into [ ] (concat (conj (subvec vet 0 pos) elem) (subvec vet
                        ↪  pos tam-vet)))))
      ] novo_vet))
```

APÊNDICE B – Código fonte da heurística inicial NEH

B.1 Arquivo NEH.clj

```
(ns NEH
  (:require [parser :as p]))

(def map-trab (p/mapeia-trabalhos))

(defn concatena-soma
  "concatena os trabalhos com o seu devido tempo de processamento total"
  []
  (into {} (zipmap map-trab (map #(reduce + ^int %) (map #(% 1) map-trab)))))

(defn ordena-trabalhos
  "ordena os trabalhos pelo tempo de processamento total em ordem decrescente"
  []
  (into [] (apply concat
                  (map pop
                       (map peek
                            (map pop
                                 (reverse (sort-by last (concatena-soma)))))))))

(def ord-trab (ordena-trabalhos))

(defn pega-vetor
  "retorna o vetor na posição i do vetor de tempos ordenados"
  [i]
  (map-trab (ord-trab i)))

(defn pega-vetor-trabalho
  "retorna o vetor do trabalho t de vetor de tempos ordenados"
  [t]
  (map-trab t))

(defn pega-trabalho
  "retorna o trabalho indicado na posição i do vetor de trabalhos ordenados"
  [i]
  (ord-trab i))

(defn elem-trab [vetor elem]
```

```
  (conj vetor elem))

(defn soma-prim-trab
  ""
  [l]
  (into [] (for [i (range (count l))]
             (reduce + ^int (subvec l 0 (+ i 1))))))

(defn calc-temp-proc
  "recebe uma lista de trabalhos e calcula o tempo de processamento de cada trabalho
   ↪  em cada maquina, retorna uma matriz com todos os tempos "
  [l]
  (let [
    tam-l (count l)
    vet [(soma-prim-trab (pega-vetor-trabalho (l 0)))]
    tempos (loop [
      inicio vet
      i 0]
      (if (= i (- tam-l 1))
        inicio
        (recur (let [
          vetor-atual (pega-vetor-trabalho (l (+ i 1)))
          tempos-parciais (loop [
            vet-aux []
            j 0]
            (if (= j p/n_maq)
              vet-aux
              (recur (if (= j 0)
                (conj vet-aux (+ ^int ((inicio i) 0) (vetor-atual 0)))
                (do
                  (if (> (vet-aux (- j 1)) ((inicio i) j))
                    (conj vet-aux (+ ^int (vet-aux (- j 1)) (vetor-atual j)))
                    (conj vet-aux (+ ^int ((inicio i) j) (vetor-atual j)))
                  ))
              ) (inc j))))
          ] (conj inicio tempos-parciais)
      ) (inc i))))
    ] tempos))

(defn calcula-makespan
  "recebe uma matriz com os tempos de processamento e retorna o makespan"
  [mat]
  (let [tam (count mat)
        makespan ((mat (- tam 1)) (- p/n_maq 1))]
    makespan))

(defn encontra-sequencia
```

```
  "retorna o melhor makespan e a melhor solução encontrada"
  []
  (let [
    vet-aux-enc [(pega-trabalho 0)]
    vet-aux-enc2 vet-aux-enc
    sequencia (loop [
      inicio [0 vet-aux-enc2]
      i 0]
      (if (= i  (- p/n_job 1))
        inicio
        (recur (let [
          makespan-melhor 9999999
          vet-enc (inicio 1)
          sequencia-parcial (loop [
            inicio-parcial [makespan-melhor vet-enc vet-enc]
            j 0]
            (if (= j (+ i 2));;permuta o trabalho j em todas as posiçoes
              inicio-parcial
              (recur (let [
                makespan-melhor (inicio-parcial 0)
                vet-aux-enc (p/add-vec vet-enc (pega-trabalho (+ i 1)) j)
                makespan-aux (calcula-makespan (calc-temp-proc vet-aux-enc))
                nova-sequencia (if (< makespan-aux makespan-melhor)
                  (let [
                    makespan-melhor makespan-aux
                    vet-aux-enc2 vet-aux-enc
                    ] [makespan-melhor vet-aux-enc2]
                  ) inicio-parcial)
                ] nova-sequencia) (inc j))))
          ] sequencia-parcial)(inc i))))
    ] sequencia))

(def NEH (encontra-sequencia))
```

APÊNDICE C – Código fonte da heurística Iteração Gulosa com Busca Local em Soluções Parciais (IG_{BLSP})

C.1 Arquivo IGBLSP.clj

```
(ns iterated-greedy
  (:require [NEH :as neh]
            [parser :as p]
            [clojure.set :as st]))

(def k_IG 2)
(def Tp 0.7)

(defn destruicao
  "recebe um vetor de n posiçoes. Retorna dois vetores resultantes da destruição, um
   ↪  com (n-K_IG) e outro com k_IG trabalhos"
  ([ent]
   (let [
         i 0
         vet ent
         posicao (take k_IG (distinct (repeatedly #(rand-int p/n_job))))
         elemento (vector (nth vet (nth posicao i)))
         vet_aux (p/remove-elem-vetor vet (nth posicao i))]
     (destruicao (inc i) vet_aux posicao elemento)))
  ([i ent pos elem]
   (let [
         vet ent
         elem_pos (if (<= (count ent) (nth pos i))
                    (- (nth pos i) i) (nth pos i))
         elemento (conj elem (nth vet elem_pos))
         vet_aux (p/remove-elem-vetor vet elem_pos)]
     (if (< i (- k_IG 1))
       (recur (inc i) vet_aux pos elemento)
       (conj (vector vet_aux) elemento)))))

(defn permuta-parcial-trabalhos
  "rearranja uma solução com (n-k_IG) trabalhos escolhendo um trabalho para permutar
   ↪  entre todos, este processo se repete (n-k_IG) vezes e no final é retornada a
   ↪  melhor permutaçao"
  [i vet vet_aux]
```

```
  (let [
        pi (second vet)
        melhor (first vet)
        pivo (nth pi i)
        del_pivo (p/remove-elem-vetor pi i)
        vet_dif (into [] (seq (st/difference (set del_pivo) (set vet_aux))))
        elem_dif (rand-nth vet_dif)
        indice_elem (.indexOf del_pivo elem_dif)
        vet_aux (conj vet_aux elem_dif)
        del_k (p/remove-elem-vetor del_pivo indice_elem)
        insere_pivo (p/add-vec del_k pivo i)
        tam (- p/n_job k_IG)
        insere_k (for [j (range tam)]
                   (p/add-vec insere_pivo elem_dif (- (- tam 1) j)))
        makespan_permutacoes (map #(neh/calcula-makespan (neh/calc-temp-proc %))
         ↪  insere_k)
        permutacoes (map vector makespan_permutacoes insere_k)
        menor (apply min-key first permutacoes)
        menor_make (first menor)
        melhor (if (< menor_make melhor) menor vet)
        ]
    (if (< i (- (- p/n_job k_IG) 2))
      (recur (inc i) melhor vet_aux)
      melhor)))

(defn buscaSolucaoParcial
  "realiza a busca parcial em uma solução enquanto nao atingir um otimo local"
  [vet flag]
  (if (= flag true)
    (let [
          pi (second vet)
          melhor (first vet)
          flag false
          novo (permuta-parcial-trabalhos 0 vet [])
          flag (if (< (first novo) melhor) true false)
          ]
      (recur novo flag))
    vet))

(defn construcao
  "reinsere os elementos que foram retirados na fase de destruicao um a um,
   ↪  inserindo-os em todas as posiçoes de forma a buscar a melhor permutação"
  [vet trabs i]
  (let [
        elem (first trabs)
        insere_k (for [j (range (- p/n_job (- k_IG i)))]
                   (p/add-vec vet elem j))
```

```
        makespan_permutacoes (map #(neh/calcula-makespan (neh/calc-temp-proc %))
         ↪  insere_k)
        permutacoes (map vector makespan_permutacoes insere_k)
        menor (apply min-key first permutacoes)
        remove (into [] (rest trabs))
        ]
    (if (< i (- k_IG 1))
      (recur (second menor) remove (inc i))
      menor)))

(defn permuta-completa-trabalhos
  "rearranja uma solução com (n) trabalhos escolhendo um trabalho para permutar
   ↪  entre todos, este processo se repete ate ser encontrado o primeiro
   ↪  melhoramento ou por (n) vezes e no final é retornada a melhor permutaçao"
  [i vet]
  (let [
        pi (second vet)
        melhor (first vet)
        pivo (nth pi i)
        del_pivo (p/remove-elem-vetor pi i)
        pos_k (rand-int (- p/n_job 1))
        k (nth del_pivo pos_k)
        del_k (p/remove-elem-vetor del_pivo pos_k)
        insere_pivo (p/add-vec del_k pivo i)
        insere_k (for [j (range p/n_job)]
                   (p/add-vec insere_pivo k j))
        makespan_permutacoes (map #(neh/calcula-makespan (neh/calc-temp-proc %))
         ↪  insere_k)
        permutacoes (map vector makespan_permutacoes insere_k)
        menor (apply min-key first permutacoes)
        menor_make (first menor)
        menor_pi (second menor)
        i (if (< menor_make melhor) (- p/n_job 2) i)
        melhor (if (< menor_make melhor) menor vet)
        ]
    (if (< i (- p/n_job 2))
      (recur (inc i) melhor)
      melhor)))

(defn buscaSolucaoCompleta
  "realiza a busca completa em uma solução enquanto nao atingir um otimo local"
  [vet flag]
  (if (= flag true)
    (let [
          pi (second vet)
          melhor (first vet)
          flag false
```

```
          novo (permuta-completa-trabalhos 0 vet)
          flag (if (< (first novo) melhor) true false)
          ]
      (recur novo flag))
    vet))

(defn temperatura
  "calcula a temperatura, ou seja, um fator para o problema estudado"
  [ ]
  (let [
        temp (reduce + (map #(reduce + %) p/ger-list))
        denominador (* (* p/n_job p/n_maq) 10)
        ]
    (/ (* Tp temp) denominador)))

(def temper (temperatura))

(defn prob-metropolis
  "calcula a probabilidade minima que uma solução deve ter para ser aprovada"
  [pi pi_2]
  (Math/exp (* (/ (double (- pi pi_2)) temper) -1)))

(defn criterio-aceitacao
  "recebe a melhor solução local, a solução corrente e a solução global. Retorna a
   ↪  nova melhor soluçao local e a global com base no criterio de aceitação"
  [pi pi_2 pi_b]
  (if (<= (first pi_2) (first pi))
    (if (< (first pi_2) (first pi_b))
      [pi_2 pi_2]
      [pi_2 pi_b])
    (if (<=
         (rand 1)(prob-metropolis (first pi_2)(first pi)))
      [pi_2 pi_b]
      [pi pi_b])))

;(def tempo (int (* p/n_job (* (double (/ p/n_maq 2)) 60))))
;(def tempo (int (* p/n_job (* (double (/ p/n_maq 2)) 120))))
(def tempo (int (* p/n_job (* (double (/ p/n_maq 2)) 240))))

(defn iterated-greedy
  "é a função principal"
  [i pi pi_b time]
  (let [
        destroi (destruicao (nth pi 1))
        destroi_vet (nth destroi 0)
        make_dest_vet (neh/calcula-makespan (neh/calc-temp-proc destroi_vet))
        destroi_resto (nth destroi 1)
```

```
        solucao_parcial (buscaSolucaoParcial [make_dest_vet destroi_vet] true)
        constroi (construcao (second solucao_parcial) destroi_resto 0)
        solucao_completa (buscaSolucaoCompleta constroi true)
        aceita (criterio-aceitacao pi solucao_completa pi_b)
        pi (first aceita)
        pi_b (second aceita)
        ]
    (if (<= (System/currentTimeMillis) time)
      (recur (inc i) pi pi_b time)
      [pi_b i])))
```

APÊNDICE D – Código fonte da meta-heurística Busca do Macaco Híbrida (BMH)

D.1 Arquivo BMH.clj

```
(ns monkey-search
  (:require [NEH :as neh]
            [parser :as p]
  ))

(def M 20)
(def v_max 4.0)
(def v_min -4.0)
(def c1 2)
(def c2 c1)

(def w0 0.9)
(def w_min 0.4)
(def beta 0.975)

(def mutacao 0.01)

(def visao 0.5)

(def salto_mortal_c -1)
(def salto_mortal_d 1)

(def iteracoes 200)
(def subidas 2000)

(def range_init 1)
(def range_velo 1)

(defn pega-elem-matriz
  "retorna o elemento na posicao [i][j] da matriz "
  [mat i j]
  ((mat i) j))

(defn valores-iniciais
  []
```

```
  (take p/n_job (repeatedly ^double #(rand range_init))))

(defn solucoes-aleatorias
  "cria M-1 soluccoes aleatórias com de acordo com n_trab entre [0, 1]"
  [m]
  (vec (map #(zipmap (shuffle (p/key-jobs)) %) (vec (for [i (range m)]
             (vec (valores-iniciais)))))))

(defn ordena-trab-aleatorios
  "recebe um vetor de trabalhos aleatorios e os ordena de acordo com o valor
   ↪  associado ao trabalho em ordem crescente "
  [l]
  (let [vet l]
    (into (sorted-map-by
            (fn [key1 key2]
              (compare [(vet key1) key1]
                       [(vet key2) key2])))
      vet)))

(defn NEH_MONKEY [] (conj [(neh/NEH 0)] (ordena-trab-aleatorios (zipmap (neh/NEH 1)
↪  (sort (valores-iniciais))))))

(defn smallest-position-value
  "aplica a funcao ordena-trab-aleatorios a todas as M solucoes geradas"
  [soluc]
  (map #(ordena-trab-aleatorios %) soluc))

(defn SPV_inicial [] (vec (smallest-position-value (solucoes-aleatorias (- M 1)))))

(defn solucao_inicial [] (conj (SPV_inicial) ((NEH_MONKEY) 1)))

(defn pega-trabalhos-SPV
  "recebe uma lista l de chaves-valores como parametro e retorna só as chaves"
 [h]
 (vec (map #(% 0) h)))

(defn trabalhos-aleatorios
  "pega cada chave-valor contido em SPV e retorna só os valores dos trabalhos de
   ↪  cada"
  [spv]
  (vec (map pega-trabalhos-SPV spv)))

(defn makespan-SPV
  "retorna o makespan de cada lista de trabalho do SPV"
  [spv]
  (vec (map #(neh/calcula-makespan (neh/calc-temp-proc %)) (trabalhos-aleatorios
   ↪  spv))))
```

```
(defn avalia-macaco
  "associa o makespan correspondende a sua lista de trabalho"
  [spv]
  (map vector (makespan-SPV spv) spv))

(defn encontra-melhor
  "retorna o melhor em uma pupoulaçao"
  [x]
    (apply min-key #(% 0) (avalia-macaco x)))

(defn velocidade-inicial
  "cria o vetor de velocidades iniciais para cada macaco no conjunto M"
  []
  (vec (for [x (range M)]
             (vec
                (take p/n_job
                  (repeatedly #(+ ^double v_min (* ^double (- v_max v_min) (rand
                   ↪  range_velo)))))))))

(defn prob-mutacao
  "verifica a probabilidade de mutacao x"
  [x]
  (> x (rand)))

(defn Xt-posicao
  "pega o vetor referente as posicoes dos macacos "
  [xt]
  (vec (for [i (range (count xt))]
              (vec (map #(% 1) (xt i))))))

(defn Xt-trabalhos
  "pega o vetor referente aos trabalhos dos macacos"
  [xt]
  (vec (for [i (range (count xt))]
              (vec (map #(% 0) (xt i))))))

(defn atualiza-peso-inercia
  "recebe como parametro um vetor com o peso inicial e a iteracao inicial e retorna
   ↪  um vetor com todos os pesos da inercia, com peso minimo 0.4, de acordo com o
   ↪  numero de iteracoes "
  [w1 i]
  (let [aux w1]
    (if (< i subidas)
      (if (< (* ^double (aux i) beta) w_min)
        (recur (conj aux w_min) (inc i))
        (recur (conj aux (* ^double (aux i) beta))(inc i)))
```

```
      aux)))

(def vetor-pesos (atualiza-peso-inercia [w0] 0))

(defn atualiza-velocidades
  "atualiza a velocidade do macaco i na dimensão j"
  [wt vij xij localij globalj]
  (let [ y (+
            (+ (* ^double wt vij) (* ^double c1 (* ^double (rand range_velo) (-
             ↪  ^double localij xij))))
            (* ^double c2 (* ^double (rand range_velo) (- ^double globalj xij))))
        ]
    (if (< y v_min)
      v_min
      (if (> y v_max)
        v_max
        y))))

(defn novas-velocidades
  "atualiza a velocidade de todos os macacos"
  [iter xt-p veloc local global]
  (vec (for [i (range M)]
            (vec (for [j (range p/n_job)]
                      (atualiza-velocidades
                        (vetor-pesos iter)
                        (pega-elem-matriz veloc i j)
                        (pega-elem-matriz xt-p i j)
                        (pega-elem-matriz local i j)
                        (global j)
                       ))))))

(defn novas-posicoes
  "atualiza as posicoes dos macacos de acordo com a velocidade e a posicao anterior"
  [xt-p veloc]
  (vec (for [i (range M)]
             (vec (for [j (range p/n_job)]
                        (+ ^double (pega-elem-matriz xt-p i j)
                            (pega-elem-matriz veloc i j)
                         ))))))

(defn concatena-chave-valor
  "recebe um conjunto chave e um conjunto valor e concatena eles"
  [chave valor]
  (for [x (range M)]
    (zipmap (chave x) (valor x))))

(defn atualiza-melhor-local
```

```
  "dado os melhores locais e os locais atuais, verifica quais sao os menores posicao
   ↪  a posicao e atualiza o melhor local"
  [melhor atual]
  (vec (for [i (range M)]
             (if (< ((atual i) 0) ((melhor i) 0))
               (atual i)
               (melhor i)))))

(defn atualiza-melhor-global
  "dado o melhor global e o global atual, verifica qual é o menor e atualiza o
   ↪  melhor global"
  [melhor atual]
  (if (< (atual 0) (melhor 0))
    atual
    melhor))

(defn mutacao-deslocamento [vet pos1 pos2]
  "aplica uma mutacão no melhor global, removendo um trabalho da posicao pos1 e
   ↪  inserindo na pos2"
  (let [
        vet-aux (vet 1)
        vet-trab (vec (map #(% 0) vet-aux))
        vet-pos (vec (map #(% 1) vet-aux))
        elem (vet-trab pos1)
        vet-trab (p/add-vec (p/remove-elem-vetor vet-trab pos1) elem pos2)
        novo-vet (vec (avalia-macaco [(ordena-trab-aleatorios (zipmap vet-trab
         ↪  vet-pos))]))
        ]
     (novo-vet 0)))

(defn verifica-mutacao
  "verifica se a mutação vai ocorrer, se sim aplica a mutação, senao nao acontece
   ↪  nada"
  [global]
  (if (prob-mutacao mutacao)
          (atualiza-melhor-global global (mutacao-deslocamento global (rand-int
           ↪  p/n_job) (rand-int p/n_job)))
          global))

(defn atualiza-global-local
  "atualiza o melhor local caso a mutação melhore o macaco"
  [global mutacao local indice ]
  (if (< (mutacao 0) (global 0))
    (p/add-vec (p/remove-elem-vetor local indice) mutacao indice)
    local))

(defn atualiza-local
```

```
  "atualiza o valor do local com o menor entre o atual e o que tinha antes"
  [anterior atual]
  (vec (for [i (range M)
             :let [
                    atual_i (atual i)
                    anterior_i (anterior i)
                  ]
           ]
         (if (< (atual_i 0) (anterior_i 0))
           atual_i
           anterior_i))))

(defn processo-subida-mutacao
  "recebe como parametros i=(iteracao atual), vij=(velocidade de cada macaco),
   ↪  xij=(posicao de cada macaco), pij=(melhor local de cada macaco), gj=(melhor
   ↪  global) e retorna as melhores posicoes locais aplicando o processo de mutacao"
  [i vij xij pij gj]
  (let [
        xij_pj (vec (map #(% 1) xij))
        xt_pos (Xt-posicao xij_pj)
        nova_vij (novas-velocidades (- i 1) xt_pos vij (vec (Xt-posicao (vec (map
         ↪  #(% 1) pij)))) (vec (map #(% 1) (gj 1))));%
        posicao (novas-posicoes xt_pos nova_vij)
        nova_xij (concatena-chave-valor (Xt-trabalhos xij_pj) posicao)
        ordena_xij (smallest-position-value nova_xij)
        local (vec (avalia-macaco ordena_xij))
        atual_local (vec (atualiza-local xij local))
        novo_pij (atualiza-melhor-local pij atual_local)
        indice_melhor_local (.indexOf novo_pij (apply min-key #(% 0) novo_pij))
        atual_global (novo_pij indice_melhor_local)
        novo_gj (atualiza-melhor-global gj atual_global)
        mutacao_gj (verifica-mutacao novo_gj)
        novo_pij (vec (atualiza-global-local novo_gj mutacao_gj novo_pij
         ↪  indice_melhor_local))
        ]
    (if (< i subidas)
      (recur (inc i) nova_vij atual_local novo_pij mutacao_gj)
      novo_pij)))

(defn escolhe-aleatorio-observa
  "recebe a iteracão atual e um vetor. Para cada elem escolhe um numero aleatorio
   ↪  entre os valores de [elem-visao, elem+visao], retorna um vetor com estes novos
   ↪  elementos se o makespan deles for menor ou igual que o makespan do vetor
   ↪  original, senao retorna o vetor original depois de um certo numero de
   ↪  tentativas de melhoramento"
  [iter x]
  (let [
```

```
      make (x 0)
      vet (x 1)
      novo_vet (vec (map #(+ ^double (rand (- (+ ^double % visao) (- ^double %
       ↪  visao))) (- ^double % visao)) ^double ((Xt-posicao [vet]) 0)))
      ordena (smallest-position-value [(zipmap ((Xt-trabalhos [vet]) 0) novo_vet)])
      novo_x (vec (avalia-macaco ordena))
    ]
    (if (or (<= ((novo_x 0) 0) make) (> iter 1000))
      (if (> iter 1000)
        x
        (novo_x 0))
      (recur (inc iter) x))))

(defn observa-salta
  "para cada macaco, chama a funcao escolhe-aleatorio-observa"
  [atual]
  (map #(escolhe-aleatorio-observa 0 (vec %)) atual))

(defn escolhe-aleat-saltomortal
  "escolhe aleatoriamente um valor entre salto_mortal_c e salto_mortal_d e monta um
   ↪  vetor com M desses valores"
  []
  (for [x (range M)] (+ ^double (rand (- ^double salto_mortal_d salto_mortal_c))
   ↪  salto_mortal_c)))

(defn media-saltomortal
  "calcula a media de cada dimensão dos macacos"
  [xij]
  (vec (map #(/ ^double % M) (apply map + ^double xij))))

(defn saltomortal
  "cada macaco da um saltomortal para uma nova posicao, de acordo com uma funcao,
   ↪  esta nova posicao será passada como parametro para a nova itreracao"
  [xij]
  (let [aleat (vec (escolhe-aleat-saltomortal))
        pivo (media-saltomortal xij)
        ]
    (vec
       (for [i (range M)]
         (vec
            (for [j (range p/n_job)]
               (+ ^double ((xij i) j)
                  (* ^double (aleat i)
                     (- ^double (pivo j)((xij i) j))))
              ))))))

(defn monkey-search
```

```
"função principal que ira chamar todas as outras funções"
[i xij global]
(let [
      y (println i "\n")
      avalia_xij (vec (avalia-macaco xij))
      climb (processo-subida-mutacao 1 (velocidade-inicial) avalia_xij avalia_xij
       ↪  (apply min-key #(% 0) avalia_xij))
      posicao (map #(% 1) climb)
      melhor (atualiza-melhor-global global (apply min-key #(% 0) climb))
      obs_salt (vec (observa-salta climb))
      melhor (atualiza-melhor-global melhor (apply min-key #(% 0) obs_salt))
      new_climb (processo-subida-mutacao 1 (velocidade-inicial) obs_salt obs_salt
       ↪  (apply min-key #(% 0) obs_salt))
      melhor (atualiza-melhor-global melhor (apply min-key #(% 0) new_climb))
      nova_posicao (vec (map #(% 1) new_climb))
      salto (saltomortal (Xt-posicao nova_posicao))
      concatena (smallest-position-value (concatena-chave-valor (Xt-trabalhos
       ↪  nova_posicao) salto))
      melhor (atualiza-melhor-global melhor (encontra-melhor concatena))
      y (println (melhor 0))
      y (println "\n----------\n")
      ]
  (if (< i (- iteracoes 1))
    (recur (inc i) concatena melhor)
    melhor)))
```