

UNIVERSIDADE FEDERAL DE UBERLÂNDIA – UFU
GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

Bruna Bernardes de Aguiar

***JOB SHOP PROBLEM* – ESTUDO DE UM MÉTODO
HEURÍSTICO PARA A SUA OTIMIZAÇÃO**

ITUIUTABA – MG

2018

Bruna Bernardes de Aguiar

JOB SHOP PROBLEM – ESTUDO DE UM MÉTODO
HEURÍSTICO PARA A SUA OTIMIZAÇÃO

Trabalho de Conclusão de Curso para a
conclusão da graduação em engenharia
de produção na Universidade Federal de
Uberlândia.

Área de concentração: Engenharia de
Produção / Pesquisa Operacional

Orientador: Jorge von Atzingen dos Reis

ITUIUTABA – MG

2018

JOB SHOP PROBLEM – ESTUDO DE UM MÉTODO HEURÍSTICO PARA SUA A OTIMIZAÇÃO

Trabalho de Conclusão de Curso
aprovado para conclusão da graduação
em engenharia de produção na
Universidade Federal de Uberlândia, pela
banca examinadora formada por:

Ituiutaba, 06 de 12 de 2018.

Prof. Dr. Jorge von Atzingen dos Reis (orientador), FACES/UFU

Prof. Dr. Lucio Abimael Medrano Castillo, FACES/UFU

Prof. Dr. Ricardo Batista Penteado, FACES/UFU

Dedico este trabalho primeiramente a Deus, pois sempre esteve comigo em todos os momentos, sendo eles bons ou ruins. Aos meus familiares que sempre estiveram ao meu lado, em principal à minha mãe e irmão e a todos os meus amigos que me deram apoio nessa jornada. A todos o meu muito obrigada.

AGRADECIMENTOS

Agradeço primeiramente a Deus por me proporcionar forças para desenvolver não só o trabalho aqui presente, mas toda a graduação e por sempre me aparar em todos os momentos.

A minha mãe e minha avó por me apoiarem em todos os desafios que passei, por sempre estarem ao meu lado, principalmente quando eu mais precisava, permanecendo comigo mesmo a distância. A todos os meus familiares, que também contribuíram com a minha formação e também agradeço os amigos que fiz durante a graduação que estiveram ao meu lado.

A Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG/UFU) por financiar o projeto aqui desenvolvido, ao meu orientador professor Dr. Jorge von Atzingen dos Reis por acreditar em mim e sempre me dar suporte para desenvolvimento do trabalho, pois seu apoio foi essencial durante esses anos.

Agradeço ao professor Dr. Lucio Abimael Medrano Castillo e ao professor Dr. Ricardo Batista Penteado por aceitarem participar da banca do trabalho aqui desenvolvido, pois mesmo não tendo muito contato nas disciplinas sei que são Docentes de extremo profissionalismo e competência, muito obrigada.

"Se você acha que pode ou que não pode fazer alguma coisa, você tem sempre razão."

- Henry Ford

RESUMO

Este trabalho busca pesquisar e desenvolver um método de otimização para problemas de programação de ordem de produção onde cada produto tem sua sequência de ordem de produção, mais especificamente o problema *Job Shop Problem* (JSP), o mesmo é classificado como um problema *NP-hard*, ou seja, não há algoritmo que encontre a solução ideal em tempo polinomial. No Planejamento e Controle da Produção (PCP) quanto mais ágil a tomada de decisão mais rápida se tem uma resposta ao cliente, aumentando de certa forma sua competitividade perante o mercado, entretanto a determinação do sequenciamento é uma tarefa complexa devido às diversas restrições que necessitam ser consideradas. O presente trabalho tem como finalidade estudar o comportamento do modelo matemático adaptado da literatura, o qual retorna à solução ótima referente ao melhor sequenciamento de tarefas a ser implementado, porém demanda um tempo elevado para retornar tal solução. Visando diminuir esse tempo foram testados métodos alternativos dentro da Pesquisa Operacional (PO) para a resolução, desenvolvendo uma heurística construtiva e posteriormente uma meta-heurística a *Variable Neighborhood Search* (VNS), que visa criar conjuntos de soluções e combiná-las entre si para chegar a melhor possível com um menor esforço computacional, ou seja, um tempo menor de resposta. O VNS necessita ainda de uma solução inicial para ser trabalhada, logo essa é a razão principal para o desenvolvimento de uma heurística construtiva, utilizar uma solução já viável para buscar uma otimizada, com um esforço computacional menor se comparado ao modelo matemático estudado. Os testes computacionais foram realizados com dados *benchmark* da literatura. Foi possível constatar que o objetivo proposto foi atingido sendo possível desenvolver uma metodologia para otimização da resolução do JSP, porém o mesmo necessita ser melhorado sendo sugerido para trabalhos futuros uma nova heurística e uma nova busca local.

Palavras-chave: Programação de Produção, Problema de Programação da Produção, Heurística Construtiva, Busca em Vizinhança Variável.

ABSTRACT

This work researching and developing one optimization method to programming problems of the production order where each product has your sequence of production order, more specific the Job Shop Problem (JSP), the same is rated as a NP-hard Problem, in other words, there are not any algorithm to found the best solution in the polynomial time. In the Planning and Production Control (PCP) as soon as possible to decision more fast the client has the answer, increasing the competition of the company in the your market, but to determine sequence of order production is a complex job because there are many restrictions need to be consider. This work has objective to study of the comportment adapted literature mathematic model, it which return the best solution of the best sequence of order production to implement, but using a long time to return the best solution. Aiming to reduce this time were tested alternative method in the Operation Research (PO) to resolve, developed one constructive heuristic and finally one Meta-heuristic the Variable Neighborhood Search (VNS), aiming to make solutions sets and combining them researching the best possible how the least computer effort, in other words, a least answer time. The VNS needs yet one initial solution to work, then it is the main reason to development of the one constructive heuristic, using one good solution to research an optimization, it with least effort computational if to compare with the mathematic model here studied. The computers tests were held with benchmark data of the literature. It was possible to verify that the proposed objective was reached, and it is possible to develop a methodology for optimizing the JSP resolution, but it needs to be improved and a new heuristic and a new local search is suggested for future work.

Keywords: Production Programming, *Job Shop Problem*, Constructive Heuristic, *Variable Neighborhood Search*.

LISTA DE ILUSTRAÇÕES

Figura 2.1 – Programação da Produção.....	23
Figura 2.2 – Exemplo Job Shop Problem.....	25
Figura 2.3 – Modelo Matemático Job Shop Problem.....	26
Figura 2.4 – Pseudocódigo Variable Neighborhood Search (VNS).	29
Figura 4.1 – Pseudocódigo heurística construtiva.	34
Figura 4.2 – Comparação da FO de cada método em cada instância de teste.	39
Figura 4.3 – Comparação do tempo de resolução em cada instância de teste.	40

LISTA DE TABELAS

Tabela 4.1 – Quantidade de máquinas e tarefas em cada instância de teste.....	32
Tabela 4.2 – Resultados obtidos com o modelo matemático adaptado.....	33
Tabela 4.3 – Resultados obtidos com a heurística construtiva.....	35
Tabela 4.4 – Calibração – Problema com 10 máquinas e 5 tarefas.	37
Tabela 4.5 – Resultados obtidos com Variable Neighborhood Search (VNS).	39
Tabela 4.6 – Resultados obtidos com 5.000 iterações e 5 estruturas de vizinhança.	41
Tabela 4.7 – Resultados obtidos com 4.000 iterações e 6 estruturas de vizinhança.	42
Tabela 4.8 – Resultados obtidos com 50 iterações sem melhoria e 6 estruturas de vizinhança.....	43
Tabela 4.9 – Percentuais relativos da função objetivo e tempo de resolução.	44
Tabela 4. 10 – FO obtida com outros tempos de resolução acima de 1 hora.....	45

LISTA DE SIGLAS E ABREVIATURAS

JSP	<i>Job Shop Problem</i>
PCP	Planejamento e Controle da Produção
PO	Pesquisa Operacional
VNS	<i>Variable Neighborhood Search</i>

SUMÁRIO

1	INTRODUÇÃO	20
1.1.	OBJETIVOS DA PESQUISA.....	20
1.1.1	<i>Objetivo Geral.....</i>	21
1.1.2	<i>Objetivos específicos.....</i>	21
1.2.	PROCEDIMENTO METODOLÓGICO	21
1.3.	RELEVÂNCIA DA PESQUISA	21
1.4.	DELIMITAÇÃO DO TRABALHO.....	21
1.5.	ESTRUTURA DO TRABALHO	22
2	FUNDAMENTAÇÃO TEÓRICA	23
2.1.	PLANEJAMENTO E CONTROLE DA PRODUÇÃO (PCP)	23
2.2.	<i>JOB SHOP PROBLEM (JSP).....</i>	25
2.2.1	<i>Modelo Matemático – Job Shop Problem.....</i>	26
2.3.	HEURÍSTICA	27
2.3.1	HEURÍSTICA CONSTRUTIVA.....	27
2.4.	META-HEURÍSTICA	28
2.4.1	<i>Variable Neighborhood Search (VNS).....</i>	28
3	MÉTODOS DE PESQUISA	30
3.1.	CARACTERIZAÇÃO DA PESQUISA	30
3.2.	TÉCNICA DE COLETA DE DADOS	30
3.3.	TÉCNICAS DE ANÁLISE DE DADOS	30
3.4.	PROCEDIMENTOS METODOLÓGICOS – ETAPAS	31
4	RESULTADOS E DISCUSSÕES	32
5	CONSIDERAÇÕES FINAIS	46
5.1.	CONCLUSÕES DO TRABALHO.....	46
5.2.	LIMITAÇÕES DO ESTUDO	46
5.3.	TRABALHOS FUTUROS	46
	REFERÊNCIAS	48
6	APÊNDICE A – ARQUIVO.CPP DA META-HEURÍSTICA VNS PROPOSTA PARA A SOLUÇÃO DO <i>JOB SHOP PROBLEM</i>.....	50

1 INTRODUÇÃO

O Planejamento e Controle da Produção (PCP) é definido como a atividade responsável por organizar os dados e as tomadas de decisões considerando todos os fatores do processo produtivo. O PCP precisa definir a melhor maneira de coordenar e aplicar os recursos da empresa de modo a atender os planos estabelecidos nos níveis estratégico, tático e operacional. Algumas das atividades mais importantes do PCP são: a previsão da demanda, o planejamento estratégico, o planejamento mestre, a programação da produção e o acompanhamento e controle da produção (TUBINO, 2009).

Dentro da programação da produção, um dos principais problemas é a alocação dos recursos para assegurar a data de conclusão dos projetos. Esse é um problema de sequenciamento da produção, conhecido na literatura como *Job Shop Problem* ou *Job Scheduling Problem* (JSP). É parte importante do Planejamento e Controle da Produção, devido às empresas necessitarem de um plano eficiente a fim de elevar sua produtividade utilizando recursos restritos, como máquina e rotinas de trabalho. Muitas empresas aplicam multas e restrições em caso de atrasos, logo, o planejamento eficiente é um fator determinante para a empresa (TUBINO, 2009).

As tarefas de um processo produtivo são independentes entre si, o que significa que cada uma possui seu próprio fluxo nas máquinas. Comum a todos os processamentos é que cada máquina pode processar uma tarefa por vez e cada tarefa só pode ser processada por uma máquina por vez. O tempo de processamento de todas as tarefas é conhecido como *makespan* e o objetivo de solucionar um JSP é tornar este tempo o menor possível (JAIN e MEERAN, 1998).

Os métodos de Pesquisa Operacional (PO) podem ser utilizados para solucionar problemas de programação como o JSP. A PO é formada por um aglomerado de técnicas de solução para os diversos tipos de problemas, a determinação da solução é baseada na complexidade do problema (TAHA, 2008).

Alguns dos métodos de PO são a programação linear, a programação inteira, a programação dinâmica, otimização em redes, a programação não linear, as heurísticas e meta-heurísticas. As heurísticas e meta-heurísticas diferentemente dos outros métodos podem não apresentar uma solução ótima, mas sim uma solução boa ou melhorada, dado que elas são utilizadas na solução de problemas que possuem modelos matemáticos muito complexos, como o caso do JSP (TAHA, 2008).

1.1. Objetivos da Pesquisa.

A seguir são expostos os objetivos que visam mostrar qual a finalidade da pesquisa, razão pela qual surgiu a temática trabalhada no objetivo geral e o detalhamento no objetivo específico.

1.1.1 Objetivo Geral

Desenvolver uma nova heurística e implementar uma Meta-heurística para a resolução do *Job Shop Problem*, visando à otimização do tempo de resposta e o *makespan*, pois não é possível encontrar a solução ótima em um tempo polinomial, sendo então testados com dados de *benchmark* da literatura.

1.1.2 Objetivos específicos

A especificidade desse trabalho se trata de implementar um modelo matemático adaptado da literatura, desenvolver uma heurística construtiva e uma Meta-heurística mais especificamente a *Variable Neighborhood Search* (VNS), todos voltados para o *Job Shop Problem*.

1.2. Procedimento Metodológico

O presente trabalho apresenta segundo a sua natureza, ser uma pesquisa aplicada, pois visa desenvolver e implementar métodos, é baseado em um problema quantitativo com objetivos exploratórios, possuindo procedimentos de Modelagem, pois um problema quantitativo utiliza unidades mensuráveis para análises e explora novos caminhos modelando o problema tratado.

Buscando sanar um problema detectado quantitativamente, através da agilidade de resposta a decisões referentes à ordem de produção para problemas de grande magnitude, onde cada tarefa tem seu próprio sequenciamento, ou seja, o *Job Shop Problem*, pesquisando metodologias dentro da Pesquisa Operacional para suprir tal demanda, sendo analisadas quantitativamente em desempenho.

1.3. Relevância da Pesquisa

A busca por um método que forneça uma excelente ordem de produção em um tempo reduzido colabora tanto com o meio empresarial quanto ao acadêmico. Diante do atual cenário de grande competitividade e busca pelo desenvolvimento, possuir tal conhecimento gera uma grande vantagem competitiva a organização, pois o grau de responsividade consequentemente aumenta. Já visando o meio acadêmico, tal pesquisa mostra que existem também outros meios para otimização da produção, abrindo o leque de opções e aprimorando os conhecimentos obtidos durante a graduação. A pesquisa ainda pode ser estendida a outras metodologias (Meta-heurísticas), instigando os discentes pela área de Pesquisa Operacional.

1.4. Delimitação do Trabalho

A pesquisa abrange duas áreas em especial, o Planejamento e Controle da Produção e a Pesquisa Operacional, utilizando-se instâncias de teste da literatura para a análise dos métodos aqui sugeridos e trabalhados, podendo ser aplicado a empresas como ferramenta de auxílio na tomada de decisão referente à programação de ordens de produção. Abordando modelos matemáticos,

heurísticas e meta-heurísticas, não sendo o intuito comparar tais métodos com outros e sim a sua eficácia para o problema tratado

1.5. Estrutura do Trabalho

No primeiro capítulo foi realizada a descrição da pesquisa proposta, assim como o problema a ser trabalho e os objetivos e demais detalhes que remetem sobre o embasamento do trabalho. No segundo capítulo se encontra o referencial teórico sobre os temas necessários para compreensão do desenvolvimento da pesquisa. O terceiro capítulo apresenta os procedimentos utilizados para atingir o objetivo proposto inicialmente no trabalho. No quarto capítulo é exposto o desenvolvimento do estudo, a aplicação da temática tratada, resultados e discussões. O quinto capítulo traz as conclusões do trabalho, limitações e sugestões para pesquisas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

Nessa seção serão abordados os temas relacionados ao planejamento e controle da produção e sua importância. Será realizado o detalhamento sobre *Job Shop Problem* e o modelo matemático proposto, será tratado também sobre o que são heurísticas e a heurística construtiva, além da temática Meta-heurística e o motivo de sua utilização e mais especificamente a *Variable Neighborhood Search* (VNS).

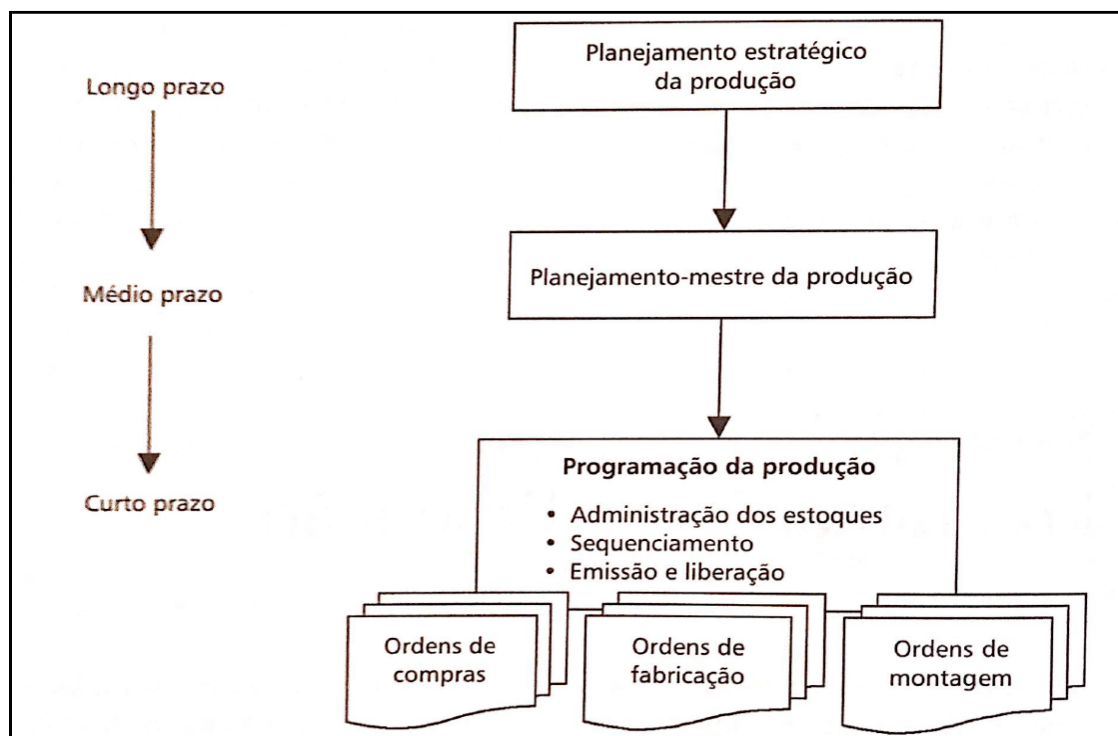
2.1. Planejamento e Controle da Produção (PCP)

Nos processos produtivos há grandes gargalos que decorre da organização e planejamento dos processos de fabricação com um todo, sendo necessário um planejamento eficiente que consiga alcançar seu objetivo final em menor tempo, com menor custo, no prazo e parâmetros de qualidade estabelecidos inicialmente.

Slack et al. (2009) analisa as tarefas do PCP como longo, médio e curto prazo. No longo prazo, há as tarefas onde se exige um maior planejamento, realizando previsões de demanda e estabelecendo os objetivos; no médio prazo, as tarefas do planejamento já são mais detalhadas; e finalmente, no curto prazo considera-se a demanda real e são determinados os recursos.

Tubino (2009) considera a previsão de demanda, o planejamento estratégico, o planejamento mestre, a programação da produção e o acompanhamento e controle da produção, como as atividades mais importantes do PCP. A Figura 2.1 representa as atividades, destacando as atividades da programação da produção.

Figura 2.1 – Programação da Produção.



Fonte: Tubino (2009).

A programação da produção é dividida em três grupos: a administração de estoques, o sequenciamento e a emissão e liberação de ordens. O foco deste trabalho é a fase de sequenciamento, na qual se gera um programa de produção que combina a fabricação e montagem dos produtos com qualidade e baixo custo (TUBINO, 2009). A Figura 2.1 representa as atividades, destacando as atividades da programação da produção.

Com um bom sequenciamento de máquinas é possível gerar inúmeros benefícios à empresa, minimizando o atraso, otimizando o tempo gasto para produção e assim atender mais pedidos, o que leva a aumentar a receita da empresa e sua credibilidade no mercado, podendo elevar a sua participação no mercado (*market share*).

O problema de programação da produção, visa realizar a entrega do produto no prazo estabelecido, otimizando o tempo de produção e a utilização de insumos, sendo o tempo de processamento e atraso a medida de desempenho principal aqui estudada.

O planejamento da produção pode ser classificado como estático ou dinâmico, a diferença entre ambos é que no planejamento estático as mudanças ocorridas durante o processo não são consideradas, já no planejamento dinâmico são considerados os imprevisto e mudanças de ordens de serviço (TUBINO, 2009).

Em sistemas considerados dinâmicos é necessário que haja uma flexibilidade do sistema de manufatura. Segundo Upton (1994), a definição de flexibilidade em tal sistema é a habilidade de se adequar através de alterações e reações com as menores penalidades possíveis em relação a alguns fatores, como o tempo, custo, esforço ou ainda desempenho do sistema.

O sistema de forma estática, assim como mencionado não leva em consideração as alterações do sistema. Porém, encontrando uma boa solução com maior agilidade, caso alguma mudança ocorra em alguma variável considerada pelo sistema, a mesma pode ser altera e a busca pela ordem de programação pode ser refeita com um tempo reduzido.

Existem ainda regras de prioridade que podem ser utilizadas para desenvolver as ordens de sequenciamento, segundo Tubino (2009) as mais utilizadas são:

- PEPS: Primeiro que entra, primeiro que sai;
- MTP: Menor tempo de processamento;
- MDE: Menor data de entrega.

Estas ordens de prioridade não são somente regras prontas e definidas, elas podem ser criadas a partir de índices importantes para a empresa e serem usadas como índices críticos.

Os sistemas de produção são sistemas computacionais que utilizam algoritmos matemáticos a fim de diminuir os tempos de transporte, de fila e de

processamento. Dentro da área de programação de produção se destaca o uso de tais sistemas, sendo eles responsáveis pelo gerenciamento das ordens de produção e requisições de compras definidas no planejamento (BREMER e LENZA, 2000).

2.2. Job Shop Problem (JSP)

O problema de sequenciamento *Job Shop* ou *Job Scheduling* é parte importante do Planejamento e Controle da Produção, pois sempre é buscado se realizar as tarefas com o mínimo de custo possível sem ter que repassar este valor aos clientes ou afetar a produção. Quando se reduz o tempo de processamento os custos tendem a reduzir, porque a quantidade produzida em um determinado período de tempo aumenta reduzindo assim os custos (variáveis e fixos), pois é possível produzir mais em menos horas trabalhadas.

Atualmente, as empresas estão buscando ainda mais diminuir o custo de produção dos seus produtos, sem que haja perda nos seus padrões de qualidade estabelecidos, portanto as pesquisas nas áreas de otimização e sequenciamento de máquinas passam a ter grande relevância, como no caso do JSP.

Segundo Pacheco e Santoro (1999) quando se existe uma série de elementos que envolvam número de máquinas, roteiro de ordens, regime de chegada de ordens, variabilidade dos tempos de processamento, entre outras características, classifica-se como sendo um problema de *Job Shop Scheduling* e para tentar resolver esse problema uma abordagem interessante é como um problema de otimização, visando ordenar as tarefas buscando a melhor eficiência possível.

Segundo Arenales et al. (2007), um *Job Shop* Clássico é um ambiente de produção com n tarefas e m máquinas, em que cada tarefa é processada nas m máquinas, de acordo com um roteiro preestabelecido. Exemplificando, considere 4 máquinas e 3 tarefas. Na figura 2.2 as matrizes M e T representam respectivamente a matriz de ordem de operações e a matriz de tempo de processamento. Sendo assim a linha 1 corresponde a primeira tarefa, a linha 2 a segunda tarefa e assim sucessivamente. Na primeira tarefa a ordem de processamento inicia-se na máquina 2 em seguida prossegue para a máquina 3, máquina 1 e finaliza na máquina 0, os tempos de processamento de cada tarefa na determinada máquina encontra-se na matriz T .

Figura 2.2 – Exemplo *Job Shop Problem*.

M=	2	3	1	0
	0	1	2	3
	3	1	2	0
T=	6	2	7	1
	2	5	9	3
	3	4	2	10

Fonte: autoria própria.

Assim no *Job Shop Problem* cada tarefa tem sua própria sequência de processamento no conjunto de máquinas com seus respectivos tempos de processamento. Outro problema clássico existente similar ao *Job Shop Problem* é o *Flow Shop Permutacional*, onde é adicionada ao problema a característica de

que para cada máquina existe sempre a mesma sequência de tarefa (NAGANO, MOCELLIN e LORENA, 2005).

2.2.1 Modelo Matemático – Job Shop Problem

Admita que n tarefas estão disponíveis para processamento inicialmente e que não é permitida a interrupção do processamento de nenhuma tarefa. Sejam os parâmetros:

- p_{ik} : Tempo de processamento da tarefa i na máquina k ;
- maq_i : Máquina que processa a i -ésima tarefa;
- M : Número suficientemente grande.

E as seguintes variáveis de decisão:

- C_{ik} : Instante de término do processamento da tarefa i na máquina k ;
- x_{ijk} : Variável binária que assume valor 1 se a tarefa i precede a tarefa j na máquina k e 0, caso contrário.

Assim, o modelo de programação matemática relativo ao *Job Shop Problem* pode ser representado pela Figura 2.3.

Figura 2.3 – Modelo Matemático Job Shop Problem.

$$\begin{aligned}
 & \min \sum_{i=1}^n C_{im} \\
 & [1] \quad C_{i,maq_1} \geq p_{i,maq_1} \quad \forall i = 1, \dots, n \\
 & [2] \quad C_{i,maq_{k+1}} \geq C_{i,maq_k} + p_{i,maq_{k+1}} \quad \forall i = 1, \dots, n \quad \forall k = 1, \dots, m-1 \\
 & [3] \quad C_{jk} \geq C_{ik} + p_{jk} - M(1 - x_{ijk}) \quad i \neq j \quad \forall i, j = 1, \dots, n \quad \forall k = 1, \dots, m \\
 & [4] \quad C_{ik} \geq C_{jk} + p_{ik} - Mx_{ijk} \quad i \neq j \quad \forall i, j = 1, \dots, n \quad \forall k = 1, \dots, m \\
 & [5] \quad C_{ik} \geq 0 \quad \forall i = 1, \dots, n \quad \forall k = 1, \dots, m \\
 & [6] \quad x_{ijk} \in [0,1] \quad \forall i, j = 1, \dots, n \quad \forall k = 1, \dots, m
 \end{aligned}$$

Erro! Fonte de referência não encontrada. Fonte: Adaptado de Arenales et al. (2007).

A função objetivo expressa a minimização do tempo de fluxo total das tarefas.

[1] – Garante que a primeira operação de cada tarefa i é completada após o respectivo tempo de processamento;

[2] – Impõe que a operação $k+1$ seja concluída depois do término da operação k e do tempo de processamento da operação $k+1$;

[3] e [4] – São restrições disjuntivas que indicam respectivamente se, na máquina k , a tarefa i precede a tarefa j , ou se a tarefa j precede a tarefa i . Caso $x_{ijk} = 1$ então se tem que: **Erro! Fonte de referência não encontrada.** e **Erro! Fonte de**

referência não encontrada., isto é, o quarto conjunto de restrições é desativado. De modo análogo, se $x_{ijk} = 0$ então **Erro! Fonte de referência não encontrada.** e **Erro! Fonte de referência não encontrada.**, isto é, o terceiro conjunto de restrições é desativado. Os dois últimos conjuntos de restrições estabelecem o tipo das variáveis;

[5] – Garante que os tempos finais de processamento das tarefas não sejam negativos o que é uma inviabilidade;

[6] – Torna a variável x_{ijk} binária.

2.3. Heurística

Segundo Glover e Kochenbergh (2003), heurísticas são estratégias de soluções que combinam qualidade de solução com esforço computacional necessário para obtê-la, ou seja, buscam uma solução satisfatória que é uma solução próxima à solução ótima, e esforço computacional aceitável que se trata utilizar em menor escala a memória do computador, reduzindo assim o tempo total de resolução.

Já as meta-heurísticas são consideradas superiores, também solucionando problemas de otimização, dado que elas controlam as heurísticas da busca local com o objetivo de melhorar o modo como determinar as soluções, explorando o espaço da vizinhança para evitar os ótimos locais (CUNHA, 2006).

Segundo Li e Chen (2010), o *Job Scheduling* é um problema de otimização combinatória, podendo ser classificado como *NP-hard*, isso significa que não existe um algoritmo que encontre a solução ideal em tempo polinomial, ou seja, o tempo de resolução cresce extremamente, ainda assim sendo bastante utilizados pelas empresas.

2.3.1 Heurística Construtiva

A heurística construtiva visa construir uma solução elemento por elemento. A forma da construção da solução varia de acordo com a função de avaliação adotada, que por sua vez depende do problema a ser trabalhado, sendo então inserido elemento por elemento na solução. No modelo clássico, os elementos ordenados para inserção na solução primeiramente costumam ser escolhidos os que fornecem maiores benefício, ou seja, uma heurística gulosa que busca pelo aumento da função objetivo que pode ser de minimização ou maximização (SOUZA, 2009).

As regras de prioridade mais utilizadas citadas por Tubino (2009), PEPS, MTP, MDE também podem ser utilizadas para determinar os elementos a serem inseridos inicialmente na solução, dentre outras para o desenvolvimento da melhor ordem de sequenciamento.

2.4. Meta-heurística

Em geral, as empresas não podem esperar muito tempo por soluções, por isto há necessidade da utilização de meta-heurísticas para encontrar soluções de qualidade em tempos aceitáveis, gerando assim melhorias contínuas. Os algoritmos agem guiando o planejador nas suas tomadas de decisões, pois estes buscam otimizar os tempos de produção.

Meta-heurísticas são métodos subsequentes com o objetivo de encontrar uma boa solução, eventualmente é possível chegar à solução ótima ou muito próxima. O funcionamento da mesma se baseia na aplicação de uma heurística submissa, a qual tem que ser moldada de acordo com o problema específico (SOUZA, 2009).

Segundo Souza (2009), a diferença principal entre heurísticas e Meta-heurísticas está no mecanismo usado para sair da emboscada dos ótimos locais que visam escapar de tais soluções ainda distantes dos ótimos globais. Elas se dividem de acordo com o mecanismo usado para buscar os espaços de soluções, podendo ser então uma busca local ou busca populacional.

Meta-heurísticas que se baseiam em uma busca local tem a exploração das soluções feita por movimentos de vizinhança, que são a geração de uma nova combinação de soluções, através de uma modificação nas soluções, sendo aplicado a cada alteração sobre uma solução corrente, gerando então uma boa solução que se trata de uma solução que se aproxima ao máximo possível da solução ótima dentro da vizinhança trabalhada (SOUZA 2009).

Ainda de acordo com Souza (2009), embora as meta-heurísticas não garantam a obtenção da solução ótima, é considerada uma boa solução aquela que chega mais próximo à solução ótima. São exemplos de métodos de busca local: a Busca Tabu, o *Simulated Annealing*, a Busca em Vizinhança Variável (*Variable Neighborhood Search*) e a *Iterated Local Search*. Já os métodos baseados em busca populacional consistem em preservar um conjunto de soluções boas e realizar a combinação entre elas, visando encontrar soluções ainda melhores, são exemplos de métodos baseados em tal busca: os Algoritmos Genéticos, os Algoritmos Meméticos e o Algoritmo Colônia de Formigas.

2.4.1 *Variable Neighborhood Search* (VNS)

O princípio do VNS visa criar conjuntos de soluções e combiná-las entre si para obter a melhor solução possível. Ele age explorando as soluções através de trocas sistemáticas de estruturas de vizinhança, e focaliza a busca em torno de uma nova solução, a qual é alcançada somente quando um movimento de melhora é identificado (MLADENOVIC e HANSEN, 1997).

As estruturas de vizinhança de uma solução são as modificações que podem ser feitas na solução atual, de tal forma a gerar uma nova solução distinta e

melhorada. Para se determinar se uma solução vizinha é uma melhora de uma anterior é utilizada a função objetivo ou função de avaliação, que tem como objetivo quantificar cada solução e selecionar as de valores superiores (REIS e CUNHA, 2010).

O VNS necessita de critérios de parada, solução inicial e uma vizinhança $N_k(s)$, sendo k um número menor que o valor determinado pelo critério e maior do que 1, os critérios podem ser tanto o tempo máximo de processamento ou o número máximo de iterações. S é o conjunto de soluções, seu vizinho vem a ser o s' que é submetido a uma busca local que resulta em uma solução s'' . Esta solução é então comparada com a atual solução, chamada de solução corrente. E então dentre estas a menor é descartada e a maior assume sua posição de solução corrente (MLADENOVIC e HANSEN, 1997).

A escolha do VNS se dá principalmente pelo fato da linha de pesquisa que originou este trabalho ter encontrado bons resultados para tais problemas utilizando essa metodologia. Além da necessidade na utilização de uma solução inicial a qual se pode utilizar a ordem de produção encontrada com a heurística construtiva, garantindo a inicialização com uma solução viável.

Figura 2.4 – Pseudocódigo *Variable Neighborhood Search* (VNS).

```

procedimento VNS()
1  Seja  $s_0$  uma solução inicial;
2  Seja  $r$  o número de estruturas diferentes de vizinhança;
3   $s \leftarrow s_0$ ;           {Solução corrente}
4  enquanto (Critério de parada não for satisfeito) faça
5       $k \leftarrow 1$ ;       {Tipo de estrutura de vizinhança corrente}
6      enquanto ( $k \leq r$ ) faça
7          Gere um vizinho qualquer  $s' \in N^{(k)}(s)$ ;
8           $s'' \leftarrow \text{BuscaLocal}(s')$ ;
9          se ( $f(s'') < f(s)$ )
10             então
11                  $s \leftarrow s''$ ;
12                  $k \leftarrow 1$ ;
13             senão
14                  $k \leftarrow k + 1$ ;
15         fim-se;
16     fim-enquanto;
17 fim-enquanto;
18 Retorne  $s$ ;
fim VNS;

```

Fonte: Souza (2009).

3 MÉTODOS DE PESQUISA

Nessa seção serão expostos quais métodos foram utilizados para o desenvolvimento do trabalho aqui delimitado e proposto.

3.1. Caracterização da pesquisa

O presente trabalho apresenta natureza aplicada, que segundo Thiollente (2009) é concentrada em volta de problemas presentes nas atividades desenvolvidas, que está empenhada no diagnóstico, identificação e busca de soluções de problemas, sendo assim, um problema quantitativo a ser estudado.

Problemas quantitativos apresentam algumas características principais como, obedecer a um plano preestabelecido com intuito de enumerar ou medir eventos, utilizar a teoria para desenvolver as hipóteses e as variáveis de pesquisa, além de examinar as relações entre as variáveis levantadas através de métodos experimentais controlados com rigor (DENZIN e LINCOLN, 2005).

O objetivo do trabalho é de fundamento exploratório, sendo um assunto ainda pouco explorado. No final de pesquisas exploratórias é possível conhecer mais sobre o assunto tratado e estar capacitado para desenvolver hipóteses, por ser muito específica dependendo muito da intuição do explorador (GIL, 2008). O procedimento para o desenvolvimento da pesquisa foi o de modelagem.

Segundo Biembengut e Hein (2003) a interação entre a matemática e a realidade pode ser levada através da modelagem visto que são conjuntos disjuntos. Modelagem matemática é a arte de transformar os problemas que pertencem à realidade em modelos matemáticos e então desenvolver sua resolução baseando-se na linguagem do mundo real.

3.2. Técnica de coleta de dados

Inicialmente, foi realizada uma pesquisa em volta de um banco de dados que possuísse dados confiáveis com soluções viáveis, visto que o foco do trabalho era o desenvolvimento de um método de otimização para o *Job Shop Problem*, não se limitando a um problema específico e sim a aplicação a todos os tipos referentes ao problema aqui tratado. Desta forma, as instâncias de teste foram retiradas do site OR-Library: <http://people.brunel.ac.uk/~mastijb/jeb/info.html> que apresenta materiais confiáveis para estudo, teste e comparações referentes a problemas clássicos.

3.3. Técnicas de análise de dados

Para a realização da análise das instâncias de testes, foi utilizado o método de comparação constante, na qual cada novo resultado obtido em determinada instância de teste é comparado com os obtidos anteriormente referentes à mesma (QUESTIONPRO, 2018).

3.4. Procedimentos metodológicos – Etapas

O desenvolvimento da pesquisa consiste nas seguintes etapas consecutivamente:

1. Estudo sobre o tema tratado e a realização do seu levantamento teórico;
2. Seleção das instâncias de teste;
3. Adaptação do modelo matemático *Job Shop Problem* (JSP) e implementação no software de otimização do GUROBI®;
4. Desenvolvimento e implementação de uma heurística construtiva para a resolução do *Job Shop Problem* (JSP);
5. Desenvolvimento da meta-heurística *Variable Neighborhood Search* (VNS);
6. Calibração da meta-heurística;
7. Análise comparativa dos métodos utilizados;
8. Análise dos resultados obtidos com o objetivo traçado para verificar se o foi atingido com eficácia.

4 RESULTADOS E DISCUSSÕES

Como descrito nos procedimentos metodológicos o estudo do tema tratado e o levantamento teórico, após tal ação o próximo passo foi realizar a seleção das instâncias de teste no banco de dados online OR-Library. As instâncias de teste foram selecionadas de acordo com o problema abordado, visando englobar desde problemas de pequena magnitude a elevadas. Foram captadas oito instâncias, mostradas na Tabela 4.1.

Tabela 4.1 – Quantidade de máquinas e tarefas em cada instância de teste.

Instância de Teste	Máquinas [Quantidade]	Tarefas [Quantidade]
1°	6	6
2°	10	5
3°	10	10
4°	15	5
5°	20	5
6°	20	15
7°	30	10
8°	50	10

Fonte: autoria própria.

Os dados *benchmark* da literatura são ordens de produção de produtos que possuem uma sequência de produção nas máquinas, logo cada máquina tem a sua ordem de processamento de tarefas, a grande questão é saber qual a melhor ordem de sequência de tarefas em cada máquina para reduzir o *makespan*. Os dados utilizados possuem inicialmente uma sequência em relação as tarefas, ou seja, qual a sequência de máquinas que cada uma deve ser processada e para a resolução aqui se determina a melhor sequência das tarefas nas máquinas.

Seguindo o planejamento foi iniciada a capacitação teórica e realização do levantamento teórico, que se encontra presente neste trabalho nas seções de introdução e revisão bibliográfica, posteriormente foi realizada a adaptação do modelo matemático *Job Shop Problem*, para a obtenção da solução ótima. Assim como o pacote de otimização utilizado para a implementação foi o GUROBI®.

Tal software possui uma opção de limitação do tempo de resolução, como trabalhado aqui com problemas de grande magnitude, foi estipulado que o tempo limite fosse de 3.600 segundos, ou seja, uma hora. A grande justificativa para tal escolha foi à viabilidade de comparações, sabendo que problemas de grande magnitude necessitam de um tempo elevado de resolução como dias para encontrar a função objetivo, e que esperar tanto tempo não se tornava viável nem

para o presente trabalho quanto para empresas que necessitam de agilidade, foi determinado esse tempo máximo com intuito de conseguir encontrar soluções boas dentro do mesmo. Com tal limitante, o modelo matemático adaptado retornou os resultados presentes na Tabela 4.2.

Tabela 4.2 – Resultados obtidos com o modelo matemático adaptado.

Problemas Máquinas X Tarefas	FO [s]	TEMPO [s]
6x6	265	2
10x5	4.832	1.100
10x10	10.602	3.600
15x5	9.006	3.600
20x5	16.651	3.600
20x15	13.367	3.600
30x10	48.266	3.600
50x10	136.381	3.600

Fonte: autoria Própria (2018).

Como estipulado no objetivo da pesquisa a finalidade seria o desenvolvimento da meta-heurística *Variable Neighborhood Search* (VNS), pois o mesmo utiliza um menor esforço computacional e retornar uma solução com menor erro em relação à solução ótima, entretanto as meta-heurísticas necessitam de uma heurística submissa, que é moldada de acordo com o problema específico, logo a heurística construtiva foi a próxima a ser implementada o seu pseudocódigo com a lógica desenvolvida encontra-se na Figura 4.1.

Figura 4.1 – Pseudocódigo heurística construtiva.

```

Heurística Construtiva ()
1  Backup dos dados de entrada
2  Selecionar a Tarefa de menor tempo de execução para que seja alocada primeiro;
3      Se (Posição de memória da Máquina está vazia) então
4          Máquina <- Tarefa;
5          Se (É a primeira Tarefa da Máquina) então
6              Tempo Inicial <- 0;
7          Senão
8              Tempo Inicial da Tarefa de menor tempo na Máquina <- Tempo
9  Final da Tarefa de menor tempo anterior na Máquina atual;
10     Tempo Final da Tarefa de menor tempo na Máquina <- Tempo
11     Inicial da Tarefa de menor tempo na Máquina + Tempo execução Tarefa de menor tempo na
12     Máquina;
13     Se (Não é a primeira Tarefa da Máquina) então
14         Tempo Inicial da Tarefa de menor tempo na Máquina <-
15     Tempo Final da Tarefa anterior na Máquina;
16     fimSe;
17     Se (Tempo Final da Tarefa atual de menor tempo na
18     Máquina anterior > Tempo Final da Tarefa anterior à Tarefa de menor tempo na Máquina atual)
19     então
20         Tempo Inicial da Tarefa de menor tempo na Máquina <- Tempo Final da Tarefa atual de
21     menor tempo na Máquina anterior;
22     Senão
23         Tempo Inicial da Tarefa de menor tempo na Máquina <- Tempo Final da Tarefa anterior a
24     Tarefa de menor tempo na Máquina atual;
25     fimSe;
26     Atualizar o valor de menor tempo para não ser selecionado novamente;
27     fimSe;
28     FO <- Soma os Tempos Finais de cada Tarefa em sua última Máquina a ser processada;
29     Fim Heurística Construtiva;

```

Fonte: autoria própria.

A escolha da heurística construtiva se deu pelo fato de ser possível determinar qual a metodologia a se utilizar para construir a solução, pois fornece assim um campo extenso de opções, e como a mesma seria utilizada somente como uma solução inicial para o desenvolvimento do VNS, possuir essa flexibilidade se tornou uma vantagem. Os resultados obtidos com a heurística construtiva encontram-se na tabela 4.3.

Tabela 4.3 – Resultados obtidos com a heurística construtiva.

Problemas Máquinas X Tarefas	FO [s]	TEMPO [s]
6x6	316	0
10x5	7.014	0
10x10	15.257	0
15x5	13.540	0
20x5	25.180	0
20x15	16.830	0
50x10	146.120	0

Fonte: autoria própria.

Possuindo os resultados da Heurística Construtiva iniciou-se o desenvolvimento da meta-heurística *Variable Neighborhood Search* (VNS), sua implementação foi com base no pseudocódigo do VNS apresentado na Figura 2.4.

Possuindo uma solução inicial de ordem de produção se realiza uma alteração na mesma, sendo então gerada a solução s' , tal solução sofre alterações de acordo com a estrutura de vizinhança que se encontra, tais alterações geram as soluções denominadas s'' .

Inicialmente é utilizada a estrutura de vizinhança 1 opt, exemplificando o que se trata tal vizinhança opt, existindo uma ordem de produção onde na máquina A deva passar inicialmente o produto 1 seguido do produto 2, produto 5 e produto 6 e na máquina B o produto 6, produto 1, produto 5 e produto 2 nessa sequência, então é realizada uma alteração onde na máquina B agora deva processar primeiro o produto 5, depois o produto 1, produto 6 e por fim o produto 2, na máquina A nada se altera. As alterações mencionadas são feitas na solução s' que geram as soluções s'' , ao realizar-se a alteração o cálculo da função objetivo é feito e então dentre as s'' desenvolvidas seleciona-se a com melhor função objetivo, caso seja melhor que a solução s' então a mesma passa a ser a nova s' , caso contrário a solução s' permanece e então avança-se para a estrutura de vizinhança 2 opt.

Na estrutura de vizinhança 2 opt o procedimento é similar ao 1 opt, porém aqui são realizadas duas alterações por solução s'' , caso se encontre uma solução melhor que a s' , então a mesma passa a ser a s' e então retorna-se para a estrutura de vizinhança 1 opt. O número total de estruturas de vizinhança é limitado pelo desenvolvedor da meta-heurística, ou seja, pode variar de acordo com as necessidades e com a calibração do programa, tema que será tratado mais a frente. Logo, essa é a lógica utilizada pela *Variable Neighborhood Search* (VNS).

Outro ponto importante do VNS é o critério de parada, ou seja, até quando serão realizadas alterações na ordem de produção para se obter a função objetivo

final, podendo ser o tempo de resolução, o número de iteração máximo ou ainda o número de iterações máximo sem melhoria na função objetivo, nesse trabalho foram utilizados como critérios de parada o número de iterações máximo e número de iterações máximo sem melhoria.

Um grande ponto a ser ressaltado se trata da lógica desenvolvida para o cálculo dos tempos finais e iniciais das tarefas a serem alocadas nas máquinas. Inicialmente foram zeradas todas as sequências de processamentos das máquinas, os tempos finais e iniciais de ambas e os auxiliares que contabilizam a sequência de alocação de tarefas e máquinas para saber quantas já foram designadas, então se partiu para os testes lógicos.

Se a tarefa fosse a primeira a ser processada na máquina e a máquina realmente fosse a primeira que deveria processar naquela tarefa: o tempo inicial de processamento da máquina passaria a ser zero e o tempo final seria o tempo de processamento daquela tarefa na máquina; então somar ia-se uma unidade a cada sequência de tarefa e sequência de máquina, pois teria sido feita uma movimentação em cada uma. Isto ocorre quando máquina ainda não foi utilizada e a tarefa está sendo processada na sua primeira etapa do processo produtivo.

Se fosse a primeira tarefa da máquina, mas não fosse à primeira máquina da tarefa: o tempo inicial da máquina seria o tempo final de processamento da tarefa na máquina anterior. Este tipo de situação ocorre quando a máquina ainda não foi utilizada em nenhuma tarefa, mas esta máquina não é a primeira etapa do processo produtivo da tarefa que está sendo processada.

Se não fosse a primeira tarefa da máquina, mas fosse à primeira máquina da tarefa: o tempo inicial da máquina seria o tempo final de processamento da tarefa anterior naquela máquina. Esta situação ocorre quando já se tem alguma tarefa alocada na máquina, mas para a tarefa em questão tal máquina é a primeira a processá-la.

Na hipótese de que não fosse a primeira tarefa da máquina nem é a primeira máquina da tarefa: seria selecionado o maior tempo final da tarefa, que seria onde ela foi processada por último e então esse passaria a ser o valor de tempo inicial daquela tarefa na máquina de sua sequência. Caso não fosse o maior tempo então, o tempo inicial passaria a ser o tempo de processamento da tarefa anterior naquela máquina; o tempo fim passaria a ser a soma do tempo inicial mais o tempo de processamento daquela tarefa na máquina atual; então somar ia-se uma unidade a sequência de tarefa e sequência de máquina, pois foram feitas movimentações em ambas.

Caso ocorra alguma inviabilidade no processo, de alocação das tarefas nas máquinas, por exemplo, a alocação de uma tarefa em na máquina antes do devido momento, então os tempos iniciais e finais são elevados para que seja feita uma nova alocação da tarefa novamente. Todo esse processo é repetido até que todas as tarefas sejam alocadas em todas as máquinas.

Após a implementação do VNS foi iniciada a sua calibração, a qual se trata de definir qual combinação dos valores das variáveis presentes nos problemas que geram uma melhor solução em um tempo reduzido.

As variáveis analisadas foram o número máximo de iterações e o número de vizinhanças, para tal determinação foi utilizado o problema de 10 máquinas e 5 tarefas, pois o mesmo foi resolvido em 1.100 segundos, não ultrapassando os 3.600 segundos estipulado como limite máximo de resolução e não sendo um problema de pequena magnitude, mostrando-se assim ser o melhor candidato para a calibração. Os dados obtidos na calibração desse problema estão presentes na Tabela 4.4.

A função objetivo encontrada nesse problema com o modelo matemático foi de 4.832 unidades com tempo de resolução de 1.100 segundos. A lógica usada para a calibração seguiu os seguintes procedimentos, selecionar um número máximo de iterações e vizinhanças para se iniciar o processo, em seguida proceder aumentando o número de iterações em mais mil iterações ou o número de vizinhanças de uma em uma unidade.

As escolhas dessas unidades são justificáveis pelos seguintes motivos: sobre o número de iterações máximos foi notado que não se havia alterações na função objetivo em menos de 1.000 iterações, sendo percebido durante a resolução inicial que foi tomada como teste para se escolher a unidade a ser aumentada, utilizando 2.000 iterações e 4 vizinhanças; sobre o número de iterações a escolha de uma unidade de vizinhança por alteração se deu pelo fato que as alterações para melhoria ou não com essa troca podem ser mais bem observadas nessa proporção coisa que se tornaria mais complexo com mais unidades a cada alteração.

A busca estava focada em encontrar uma função objetivo próximo à ótima obtida com o modelo matemático, porém com um tempo reduzido, logo a lógica seguiu essa premissa, buscando a combinação ideal do número de iterações e número de vizinhanças, reduzindo e aumentando ambas até alcançar objetivo.

Tabela 4.4 – Calibração – Problema com 10 máquinas e 5 tarefas.

Nº Máximo de Iterações	Nº Vizinhança	Função Objetivo [s]	Tempo de Resolução [s]
2.000	4	5.280	19
3.000	4	5.280	28
2.000	5	5.275	25
3.000	5	5.275	36
4.000	5	5.155	44
100.000	4	5.211	908
11.027	4	5.149	101
100.000	3	5.506	701

Fonte: autoria própria.

Com aumento para 3.000 iterações o tempo de resolução aumentou mantendo a mesma função objetivo, o que levou a reduzir o número máximo de

iterações para 2.000 e aumentar o número de vizinhança para 5, isso gerou uma função objetivo melhor.

O tempo de resolução ainda era elevado quando comparado ao da combinação inicial da calibração, levando a aumentar o número máximo de iterações novamente para 3.000 e mantendo o número de vizinhança em 5, com essa alteração a função objetivo não foi melhorada e o tempo de resolução foi aumentado. O passo seguinte foi aumentar o número de iterações para 4.000 mantendo número de vizinhança em 5, isso gerou uma função objetivo mais próxima da ótima 5.155 unidades.

Para observar em qual iteração a função objetivo se aproximava da ótima ainda com 4 estruturas de vizinhança foi implementado o número máximo de iterações de 100.000, ao decorrer da resolução se constatou que a função objetivo se tornava melhor que a encontrada no teste anterior na iteração 11.027, logo foi implementado a combinação de 4 vizinhanças e o número 11.027 de iteração máxima com isso o tempo de resolução passou a ser 101 segundos.

O mesmo teste foi realizado com o número máximo de iterações sendo 100.000 e 3 estruturas de vizinhança, porém a função objetivo não atingiu a obtida no teste anterior, confirmando que três estruturas de vizinhança não atendiam ao objetivo proposto.

Mesmo que o teste de 4 estruturas de vizinhança e o número 11.027 de iteração máxima tenha gerado uma função objetivo de 5.149 unidades o tempo de resolução foi de 101 segundos, quando comparado com a outra combinação de 5 estruturas de vizinhança e o número 4.000 de iteração máxima que gerou uma função objetivo de 5.155 unidades com tempo de resolução de 44 segundos, é possível comprovar que a melhora foi pouco significativa considerando o aumento necessário no tempo de resolução. Desta forma, foi selecionada a combinação de 4.000 iterações e 5 vizinhanças, pois gerou uma função objetivo boa e em um tempo menor. Os resultados obtidos com o VNS calibrado encontram-se na Tabela 4.5 a seguir.

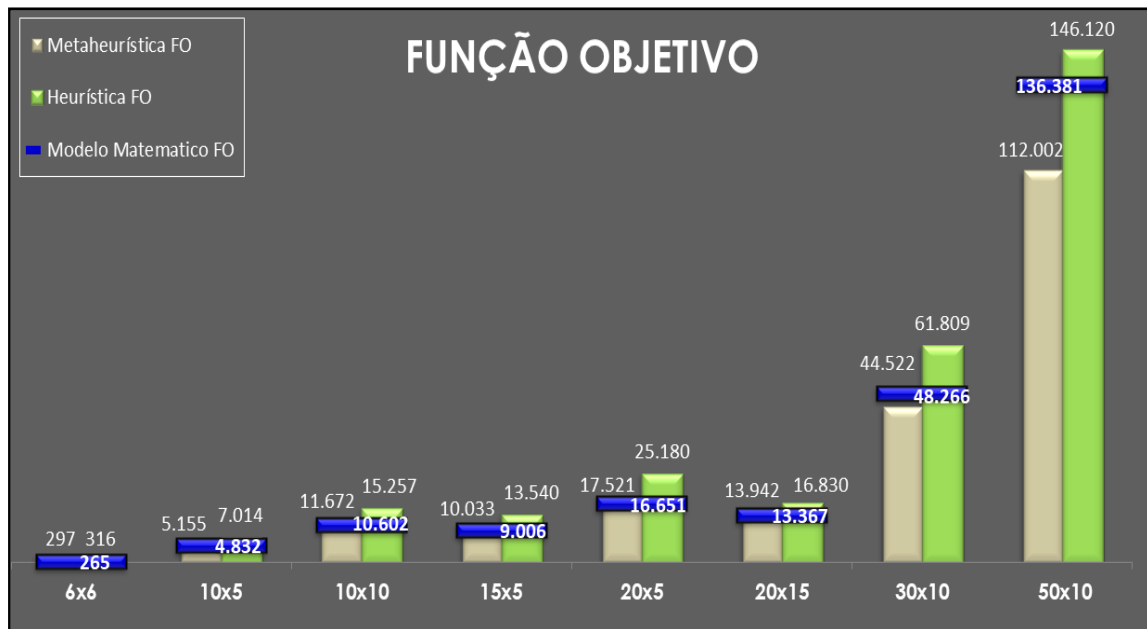
Tabela 4.5 – Resultados obtidos com *Variable Neighborhood Search* (VNS).

Problemas Máquinas X Tarefas	FO [s]	TEMPO [s]
6x6	297	14
10x5	5.155	44
10x10	11.672	178
15x5	10.033	199
20x5	17.521	575
20x15	13.942	3.765
30x10	44.522	8.538
50x10	112.002	14.400

Fonte: autoria própria.

Possuindo todos os resultados a análise comparativa foi iniciada, para uma melhor análise foram gerados gráficos dos resultados obtidos com cada método desenvolvido e aplicado. Para comparação da função objetivo de cada método foi gerado o gráfico da Figura 4.2.

Figura 4.2 – Comparação da FO de cada método em cada instância de teste.



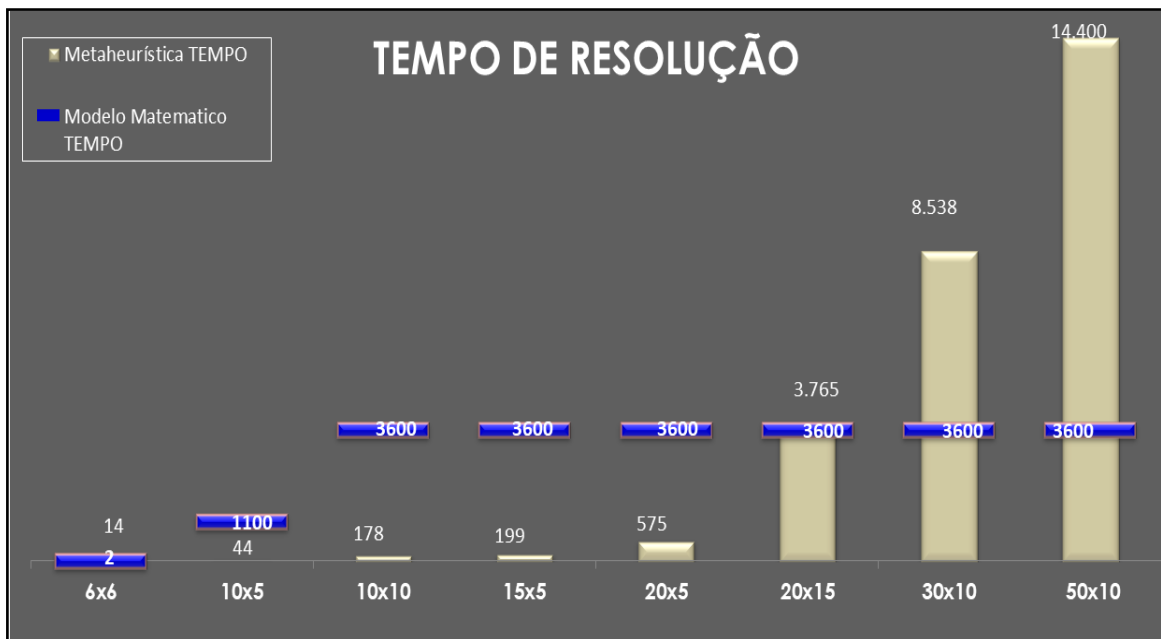
Fonte: autoria própria.

Os resultados obtidos com o modelo matemático foram utilizados como modelo de solução ótima para comparação com os outros dois métodos. É notório que a heurística ultrapassa significativamente a função objetivo obtida com o modelo matemático e como mencionado à mesma só foi desenvolvida com a finalidade de ser utilizada como solução inicial do VNS, logo ter uma FO não satisfatório era esperado. Sobre a meta-heurística, em seis instâncias de teste a

mesma chegou muito próximo ao valor obtido com o modelo matemático e em dois casos a mesma ficou abaixo, esse fato ocorreu, pois foi limitado o tempo de resolução do modelo matemático em 3.600 segundos.

Assim como com a função objetivo foi gerado um gráfico para a comparação do tempo de resolução de cada método na Figura 4.3. Os resultados obtidos com a heurística construtiva foram retirados, pois a mesma obteve tempo de resolução para todas as instâncias de 0 segundos, além de não ser o objetivo principal.

Figura 4.3 – Comparação do tempo de resolução em cada instância de teste.



Fonte: autoria própria.

É importante observar que a meta-heurística é mais lenta do que o modelo matemático em três das oito instâncias de teste. Com tais resultados obtidos com a calibração outras hipóteses foram levantadas sendo elas, caso o número máximo de iterações tivesse sido aumentado em mais 1.000 talvez fosse possível encontrar alguma melhoria, ou então se tivesse sido aumentado o número de iterações em mais uma unidade, pensando nisso foram realizados tais testes. A combinação de 5.000 como número máximo de iterações e 5 estruturas de vizinhança sendo um teste e o outro com 4.000 como número máximo de iterações e 6 estruturas de vizinhança.

Outra opção na busca pelo alcance do objetivo traçado na pesquisa foi o de utilizar outra solução inicial que não fosse a obtida pela heurística construtiva, então surgiu outro teste que se trata de utilizar o GUROBI® para encontrar a solução inicial, estipulando 60 segundos como tempo limite para a resolução com o software, pois a heurística utiliza um tempo extremamente reduzido para obter uma solução como foi exposto anteriormente.

Foram então realizados testes com as duas propostas de aumentar o número de iterações para 5.000 e permanecer com 5 estruturas de vizinhança e a de aumentar o número de estruturas de vizinhanças para 6 e permanecer com 4.000

iterações, ambas iniciando com a solução inicial gerada pela heurística construtiva e pela solução gerada dentro de 60 segundos pelo pacote de otimização GUROBI®.

Tabela 4.6 – Resultados obtidos com 5.000 iterações e 5 estruturas de vizinhança.

Problemas	VNS		GUROBI®	
	FO [s]	TEMPO RESOLUÇÃO [s]	FO [s]	TEMPO RESOLUÇÃO [s]
6x6	295	18	265	78
10x5	5.149	62	4.870	118
10x10	11.335	208	10.685	275
15x5	9.397	218	9.341	266
20x5	17.696	670	15.960	784
20x15	13.965	4.950	15.200	13.871
30x10	46.240	10.894	11.000.000	12.240
50x10	116.164	62.022	34.000.000	89.007

Fonte: autoria própria.

Pode-se observar que houve melhorias na função objetivo se comparado os resultados obtidos com os da calibração de 5.000 iterações e 5 estruturas de vizinhança para os problemas de magnitude menor, mas isso não ocorreu em problemas de maiores dimensões.

Quando são analisados os resultados obtidos com a solução inicial da heurística construtiva com os obtidos com a solução gerada pelo GUROBI® em 60 segundos, nota-se que existem melhoras na função objetivo para problemas também de magnitude menor, porém em problemas com maior complexidade a solução objetivo é aumentada consideravelmente, além de demanda um tempo de resolução ainda maior, mostrando a inviabilidade para esse caso de 5.000 iterações e 5 estruturas de vizinhança.

O próximo teste foi aumentar o número de estruturas mantendo o número de iterações fixo, sendo então 4.000 iterações e 6 estruturas de vizinhança, os resultados obtidos encontram-se na Tabela 4.7.

Tabela 4.7 – Resultados obtidos com 4.000 iterações e 6 estruturas de vizinhança.

Problemas	VNS		GUROBI®	
	FO [s]	TEMPO RESOLUÇÃO [s]	FO [s]	TEMPO RESOLUÇÃO [s]
6x6	301	15	265	74
10x5	5.629	53	4.906	114
10x10	11.507	188	10.685	245
15x5	9.855	198	9.619	281
20x5	18.337	683	15.289	777
20x15	14.018	4.733	142.108	4.962
30x10	48.682	10.601	80.754	21.786
50x10	189.666	12.789	203.600	30.634

Fonte: autoria própria.

Os resultados obtidos com tal alteração não geraram melhoras significativas a função objetivo se comparada os resultados obtidos no teste anterior e ainda utilizam um tempo maior de resolução. Os resultados obtidos com a solução inicial gerada pelo GUROBI® em alguns casos são melhores do que a gerada pela heurística construtiva, mas tal fato não se estende a todos os problemas, sendo em alguns extremamente maiores, mostrando também para esse caso sua ineficiência.

Ressaltando que o objetivo traçado é encontrar uma solução próxima a ótima em um tempo inferior ao utilizado pelo modelo matemático, outro teste foi realizado, sendo a alteração do critério de parada. O novo critério de parada visa parar a busca pela solução quando forem realizadas até 50 iterações sem nenhuma melhoria na função objetivo e 6 estruturas de vizinhança para tal teste também foi realizada a comparação com a solução inicial sendo encontrada pelo pacote de otimização GUROBI® dentro de 60 segundos de resolução. Os resultados obtidos com tal teste encontram-se na Tabela 4.8.

Tabela 4.8 – Resultados obtidos com 50 iterações sem melhoria e 6 estruturas de vizinhança.

Problemas	VNS		GUROBI®	
	FO [s]	TEMPO RESOLUÇÃO [s]	FO [s]	TEMPO RESOLUÇÃO [s]
6x6	306	0	265	60
10x5	5.629	2	4.906	61
10x10	12.031	8	10.685	62
15x5	9.936	19	9.871	67
20x5	19.672	76	16.389	108
20x15	14.021	1.146	13.878	647
30x10	46.682	8.456	50.754	7.998
50x10	130.666	12.601	140.600	11.404

Fonte: autoria própria.

Com tais resultados é notório que os mesmos são melhores do que os obtidos nos testes anteriores aqui analisados, porém nota-se que iniciando com a solução do GUROBI® o tempo de resolução tende a aumentar consideravelmente quando comparado que utilizou a solução inicial da heurística. Os resultados obtidos com a solução inicial do GUROBI® são melhores para problemas de magnitude menor, porém para os demais problemas ele tende a perder eficiência a medida que a complexidade aumentada.

Para um análise completa de todos os testes com exceção dos testes com solução inicial gerada pelo GUROBI®, pois o mesmo aumenta significativamente o tempo de resolução e não tal fator não é o objetivo do trabalho, foram realizados os cálculos de todos os erros percentuais relativos os valores obtidos com o modelo matemático e também a redução ou aumento percentual do tempo de resolução, a fórmula para tais cálculos se encontra abaixo e os percentuais obtidos estão na tabela 4.9.

$$\frac{[(\text{Novo valor} - \text{Valor antigo do Modelo Matemático}) / \text{Valor antigo do Modelo Matemático}] \times 100}{\text{Variação percentual} \times 100}$$

Tabela 4.9 – Percentuais relativos da função objetivo e tempo de resolução.

Probl.	4.000 Iterações e 5 Estruturas de Vizinhança		5.000 Iterações e 5 Estruturas de Vizinhança		4.000 Iterações e 6 Estruturas de Vizinhança		50 Iterações sem melhoria e 6 Estruturas de Vizinhança	
	FO [%]	TEMPO RESOLUÇÃO [%]	FO [s]	TEMPO RESOLUÇÃO [%]	FO [s]	TEMPO RESOLUÇÃO [%]	FO [s]	TEMPO RESOLUÇÃO [%]
6x6	12%	600%	11%	800%	14%	650%	15%	-100%
10x5	7%	-96%	7%	-94%	16%	-95%	16%	-100%
10x10	10%	-95%	7%	-94%	9%	-95%	13%	-100%
15x5	11%	-94%	4%	-94%	9%	-95%	10%	-99%
20x5	5%	-84%	6%	-81%	10%	-81%	18%	-98%
20x15	4%	5%	4%	38%	5%	31%	5%	-68%
30x10	-8%	137%	-4%	203%	1%	194%	-3%	135%
50x10	-18%	300%	-15%	1623%	39%	255%	-4%	250%

Fonte: autoria própria.

Os resultados mostram que uma opção seria o critério de parada com 50 iterações sem melhoria e 6 estruturas de vizinhança, ainda que tenha obtido percentuais relativos da função objetivo maiores em alguns casos o que reduz significativamente o tempo de resolução, entretanto, analisando os resultados de todos os testes realizados no presente trabalho, se percebe que o VNS precisa de alguma estrutura de vizinhança que explore melhor o espaço de soluções, simplesmente aumentar ou reduzir o número de vizinhanças não resolverá o problema. A busca local apresenta-se lenta e possivelmente uma busca local decomposta possa ajudar.

Reduzir o número de iterações reduz o tempo para a resolução do problema, mas piora a solução obtida. A mudança o critério de parada sendo o número de iterações sem melhora em vez de número de iterações máximas reduz o tempo, mas piora a qualidade da solução nas instâncias de maiores dimensões. O VNS está funcionando bem e consegue melhorar bastante a solução inicial, ao partir de uma solução inicial de melhor qualidade consegue chegar mais próximo do valor da solução ótima. Talvez uma nova heurística construtiva para a solução inicial possa ser uma boa melhoria.

A potência da máquina utilizada não aumenta tanto sua eficiência, pois as heurísticas e meta-heurísticas são metodologias robustas já utilizando sua capacidade ao máximo. Foi utilizado 1 hora como tempo máximo de resolução, então levanta-se a hipótese sobre a confiabilidade dos resultados, portanto foram realizados teste para determinar a variação da função objetivo de acordo com o tempo de resolução utilizado foram testados os tempos de 1 hora e 8 horas e 24 horas gerando os seguintes dados.

Tabela 4. 10 – FO obtida com outros tempos de resolução acima de 1 hora.

Problemas	Tempo de Resolução : 1 hora	Tempo de Resolução : 8 horas	Tempo de Resolução : 24 horas
6x6	306	306	306
10x5	5.629	5.622	5.627
10x10	12.031	12.021	12.019
15x5	9.936	9.999	9.947
20x5	19.672	19.613	19.690
20x15	14.021	14.501	14.142
30x10	44.583	45.003	44.603
50x10	120.906	122.003	120.106

Fonte: autoria própria.

Com os resultados obtidos é possível confirmar a robustez das metodologias, garantindo que a utilização de uma hora não influencia em grande proporção nas conclusões encontradas.

5 CONSIDERAÇÕES FINAIS

Nessa seção serão expostas as considerações finais do trabalho desenvolvido, expondo as conclusões obtidas, as limitações do estudo e também os possíveis trabalhos a serem desenvolvidos futuramente.

5.1. Conclusões do trabalho

O objetivo final do trabalho traçado seria desenvolver um método de solução para o *Job Shop Problem* que se utiliza um menor esforço computacional e retorne-se uma solução menor em relação à solução ótima. Para obter tão solução foi traçada então a estratégia de desenvolver uma heurística para obter a solução inicial e então implementar a Meta-heurística *Variable Neighborhood Search* (VNS) comparando os resultados com o obtido pelo modelo matemático que gera a solução ótima.

Tais estratégias foram aplicadas e estudadas, cumprindo o objetivo traçado, entretanto, assim como citado o VNS necessita de algumas melhorias a serem pesquisadas. Assim como exposto nos resultados e discussões o critério de parada de 50 iterações sem melhoria e 6 estruturas de vizinhança gerou uma redução no tempo de resolução em relação ao modelo matemático em 6 das 8 instâncias de testes, com uma média de aumento na função objetivo também em 6 das 8 instâncias, porém esse aumento percentual relativo representa em grande maioria menos de 19% o qual para tais problemas mostra uma melhora significativa.

Podemos então concluir que o objetivo foi traçado com sucesso e que mesmo o resultado não sendo completamente o esperado, os resultados gerados são bons e de relevância para a resolução de tais problemas e de avanço para a linha de pesquisa *Job Shop Problem*.

5.2. Limitações do estudo

Para problemas de instância maior o tempo de resolução era elevado e ocasionalmente os resultados não eram os esperados, na grande maioria dos testes divergências o que reiniciava a busca para alcançar o objetivo traçado, com isso grande parte do tempo de desenvolvimento do trabalho foi consumido gerando uma limitação para o desenvolvimento de mais testes. Computadores mais potentes não geram grandes efeito no tempo de resolução, logo a máquina utilizada não foi um limitante para a pesquisa.

5.3. Trabalhos futuros

Como trabalhos futuros é pressuposto que com os resultados expostos no presente trabalho um caminho que deva gerar bons resultados, assim como previamente citado deve ser o desenvolvimento de uma nova heurística para a obtenção de uma solução inicial melhor e um busca local que consiga ser mais eficiente, por exemplo, uma busca local decomposta que consegue explorar as soluções vizinhas a solução s' que podem ser alcançadas através da recondução

das tarefas escolhidas de forma aleatória na solução s' , garantindo assim uma liberdade e exploração do espaço de soluções viáveis ainda maior (REIS, 2013) e tal trabalho a aplicação da busca local decomposta foi implementada para problemas de transporte. A proposta referente a utilização de uma nova busca local se dá pelo fato de que a limitação de testes computacionais no presente trabalho foi causada principalmente pelo tempo de resolução ainda alto para instâncias de testes maiores e uma busca decomposta poderá agilizar tal tarefa.

Outra proposta de continuidade que pode ser explorada é a utilização de outras estruturas de vizinhança, um exemplo seria uma estrutura de reconstrução a qual poderia reordenar a produção nas máquinas de acordo com um critério estabelecido pelo desenvolvedor com o objetivo de melhorar a função objetivo, sendo devidamente estudado qual seria tal fator que poderia contribuir para otimizar o esforço computacional (REIS, 2013). No trabalho seguido como referência aqui a aplicação foi realizada em problemas de logística e o intuito aqui é realizar a implementação de tais trabalhos no PCP.

REFERÊNCIAS

- ARENALES, M. et al. (2007). **Pesquisa Operacional**. Rio de Janeiro: Elsevier
- BIEMBENGUT, M. S. e Hein.N. **Modelagem Matemática no ensino**.São Paulo: Contexto, 2003.
- BREMER, C. F.; LENZA, R. P. **Um modelo de referência para gestão da produção em sistemas de produção assembly to order – ATO e suas múltiplas aplicações**. Gestão e Produção, v. 7, p. 268-282, dez 2000.
- CUNHA, C. B. (2006). **Contribuição à Modelagem de Problemas em Logística e Transportes**. Tese (livre docência), Escola Politécnica 1. ed. São Paulo: Universidade de São Paulo. 315 p
- DENZIN, N. K. e LINCOLN, Y. S. **Handbook of Qualitative Research**. Thousand Oaks: Sage, 2005
- GIL, Antonio Carlos. **Como elaborar projetos de pesquisa**. 5. ed. São Paulo: Atlas, 2008.
- GLOVER, F.; KOCHENBERG, G. A. (2003). **Handbook of Metaheuristics. International Series in Operations Research e Management Series, Kluwer's International Series**, Stanford University, 556p.
- JAIN, A S, MEERAN, S. **Deterministic job-shop scheduling: past, present and future**. Departament of Applied Physics and Eletronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, UK, DD1 4HN. 1998.
- Li, Y. e Chen, Y., **A genetic algorithm for job-shop scheduling**. Journal of Software, 5(3): 269-274 (2010).
- MLADENOVIC, N. E HANSEn, P.(1997), **Variable neighborhood search**. Computers and Operations Research, 24, 1097-1100.
- Nagano, M. S.; Moccellini, J. V. e Lorena, L. A. N. **Programação da produção flow shop permutacional com minimização do tempo médio de fluxo**. In Anais do 36º Simpósio Brasileiro Pesquisa Operacional, São José Del-Rei, 2004.
- PACHECO, R. F. ; SANTORO, M. C. . **A adoção de modelos de scheduling no Brasil: deficiências no processo de escolha**. Gestão e Produção, São Carlos/SP, v. 8, n. 2, p. 128-138, 2001.
- QUESTIONPRO (2018). **Conselhos para análises de dados de uma pesquisa quantitativa**. Disponível em: <<https://www.questionpro.com/blog/pt-br/conselhos-para-analises-de-dados-quantitativo>>. Acessado em 06 de Outubro de 2018.
- REIS, J. A. **Meta-Heurísticas baseadas em Busca em Vizinhança Variável Aplicadas a Problemas de Operação de Transportes**. 2013. Tese (Doutorado

em Engenharia de Produção) - Escola Politecnica, Universidade de São Paulo, São Paulo.

REIS, J. A. E CUNHA, C. B. (2010) **Uma heurística baseada em VNS para o problema bidimensional de binpacking com frota heterogênea**. In: XVI PANAM, 15-18 Julho, 2010 – Lisboa, Portugal.

SLACK, N.; CHAMBERS, S.; JOHNSTON, R. **Administração da produção**. 3. ed. – São Paulo : Atlas, 2009.

SOUZA, Jamilson Freitas Souza, (2009). **Inteligência Computacional para Otimização**. Disponível em: <<http://www.decom.ufop.br/prof/marcone>>. Acessado em 20 de Julho de 2018.

TAHA, Hamdy A. (2008) **Pesquisa operacional: uma visão geral** / Hamdy A. Taha; tradução Arlete Simille Marques; revisão técnica Rodrigo Arnaldo Scarpel. – São Paulo: Pearson Prentice Hall.

THIOLLENT, M. **Metodologia da pesquisa-ação**. São Paulo: Cortez, 2009.

TUBINO, D. F. **Planejamento e controle da produção: teoria e prática** / Dalvio Ferrari Tubino. - 2. ed. – São Paulo: Atlas, 2009.

UPTON, D. **The management of manufacturing flexibility**. California Management Review, Winter, p.72-89, 1994.

6 APÊNDICE A – ARQUIVO.CPP DA META-HEURÍSTICA VNS PROPOSTA PARA A SOLUÇÃO DO JOB SHOP PROBLEM

```
//-----
//Inclusao das bibliotecas
//-----
#include "gurobi_c++.h"
#include <sstream> //cin, cout, ostringstream
#include <fstream> //ifstream, ofstream
#include <vector> //vector
#include <time.h> //clock_t
#include <limits.h> //INT_MAX
#include <stdlib.h> //rand
using namespace std;

//-----
//Declarações das Constantes
//-----
#define M 99999 //número suficientemente grande
#define MAX_JOB 50 //número máximo de tarefas
#define MAX_MAQ 10 //número máximo de máquinas
#define ITERMAX 4000 //número máximo de iterações ou 50 se forem
// iterações sem melhora (iter=0; na linha 997)
#define KMAX 6 //número máximo de estruturas de vizinhança

//-----
//Declarações das variáveis globais
//-----
class jobs
{
public:
    int maq[MAX_MAQ]; //máquina que processa a tarefa i na
// máquina k [njob][nmaq];
    int tp[MAX_MAQ]; //tempo de processamento da tarefa i na máquina
// k;
    int tp2[MAX_MAQ]; //o mesmo tempo de processamento mas ordenado
// por máquina k;
};

vector <jobs> job;
float FO, FOs0, FOs1, FOs2, FOs3;
float respostaX[MAX_JOB][MAX_JOB][MAX_MAQ]; //Resultado da variável de
// decisão X
float respostaC[MAX_JOB][MAX_MAQ]; //Resultado da variável de
// decisão C
int njob; //número de jobs
int nmaq; //número de máquinas

time_t t_ini, t_fim; //contadores de tempo computacional
```

```

double tempo;                                //contadores de tempo computacional

class machines
{
    public:
        int tarefa[MAX_JOB];    //tarefa i que é processada na máquina k;
        int tempoini[MAX_JOB];  //tempo de processamento inicial da tarefa i
        na máquina k;
        int tempofim[MAX_JOB];  //tempo de processamento final da tarefa i na
        máquina k;
};

vector <machines> maquina0;    //solução s0
vector <machines> maquina;    //solução s
vector <machines> maquina1;   //solução s1
vector <machines> maquina2;   //solução s2
vector <machines> maquina3;   //solução s3

//-----
//Declaração das funções
//-----
void buscalocal();
void calculafo(int s);
void calculatempo(int s);
void formatacao();
void heuristicaconstrutiva();
void imprime(int s);
void jobshopgurobi(int s);
void leia();
int menu();
void troca1opt(int i, int j, int maq, int s);
void troca2opt(int s);
void troca3opt(int s);
void troca4opt(int s);
void troca5opt(int s);
void troca6opt(int s);
void vns();

//-----
//Programa principal
//-----
int main()
{
    int escolha;
    do{
        escolha=menu();
        switch (escolha)
        {
            case 1:
                leia();

```

```

        jobshopgurobi(0);
        imprime(0);
        break;

    case 2:
        leia();
        heuristicaconstrutiva();
        imprime(1);
        break;

    case 3:
        leia();
        heuristicaconstrutiva();
        vns();
        imprime(1);
        break;

    case 4:
        leia();
        jobshopgurobi(1);
        formatacao();
        vns();
        imprime(1);
        break;
    }
}while(escolha);
return 0;
}

//-----
//menu: menu de escolha
//-----
int menu()
{
    int choice;
    do{
        cout<<"\n-----\n";
        cout<<"1- ProgramaÃ£o Linear\n";
        cout<<"2- HeurÃstica Construtiva\n";
        cout<<"3- Busca em VizinhanÃ§a VariÃvel\n";
        cout<<"4- PL com VNS\n";
        cout<<"0- Sair.\n";
        cout<<"-----\n";
        cin>>choice;
    }while(choice<0||choice>4);
    return choice;
}

//-----
//FunÃ£o leia: LÃª o arquivo de entrada gurobi109.entrada.txt

```

```
//-----
void leia()
{
    ifstream origem ("gurobi109.entrada.txt");
    if (!origem)
        cerr<< "\nErro ao abrir o arquivo gurobi109.entrada.txt\n\n";

    jobs dado;
    job.resize(0);
    origem>> njob;
    origem>> nmaq;
    for (int i=0; i<njob; i++)
    {
        for (int k=0; k<nmaq; k++)
        {
            origem>> dado.maq[k];
            origem>> dado.tp[k];
        }
        job.push_back(dado);
    }
    //ordenar os tempos de processamento por máquina
    for (int i=0; i<njob; i++)
        for (int k=0; k<nmaq; k++)
            job[i].tp2[job[i].maq[k]]= job[i].tp[k];

    machines dado2;
    maquina.resize(0);
    for (int k=0; k<nmaq; k++)
    {
        for (int i=0; i<njob; i++)
        {
            dado2.tarefa[i]=-1;
            dado2.tempoini[i]=0;
            dado2.temposfim[i]=0;
        }
        maquina.push_back(dado2);
    }
}

//-----
//      min sum from {i=1} to n C_{i m} newline C_{i, maq_1} geslant p_{i, maq_1}
~~~ forall i=1,...,n newline C_{i, maq_{k+1}} geslant C_{i, maq_k} + p_{i,
maq_{k+1}} ~~~ forall i=1,...,n ~~~ forall k=1,...,m-1 newline C_{jk} geslant C_{ik} +
p_{jk} - M (1-x_{ijk}) ~~~ i<>j forall i,j=1,...,n ~~~ forall k=1,...,m newline C_{ik}
geslant C_{jk} + p_{ik} - M x_{ijk} ~~~ i<>j forall i,j=1,...,n ~~~ forall k=1,...,m newline
C_{ik} geslant 0 ~~~ forall i=1,...,n ~~~ forall k=1,...,m newline x_{ijk} in [0,1] ~~~
forall i,j=1,...,n ~~~ forall k=1,...,m
//-----
void jobshopgurobi(int s)
{
```

```

//-----
//GUROBI
//-----
time(&t_ini);
GRBEnv* env = 0;
GRBVar** c = 0;
GRBVar*** x = 0;

try
{
    cout<<"\n-----\n\n";
    cout<<"\nIniciando a resolu  o do problema.\n\n";
    env = new GRBEnv(); //inicia ambiente gurobi
    GRBModel model = GRBModel(*env); //cria um modelo
    model.set(GRB_StringAttr_ModelName, "Job.Shop");
    switch (s)
    {
        case 0:
            model.getEnv().set(GRB_DoubleParam_TimeLimit, 60); //Tempo limitado a
3.600s
            break;

            case 1:
                model.getEnv().set(GRB_DoubleParam_TimeLimit, 60); //Tempo limitado a
60s
                break;
    }
    model.getEnv().set(GRB_DoubleParam_Heuristics, 0.25); //Heur stica utilizada
durante 5% do tempo computacional

    //-----
    //Dados de entrada
    //-----
    c = new GRBVar* [njob];
    for(int i= 0; i<njob; i++)
    {
        c[i] = model.addVars(nmaq, GRB_INTEGER);
        model.update();
        for (int k= 0; k< nmaq; k++)
        {
            ostringstream vname;
            vname << "c" << i << k;
            c[i][k].set(GRB_DoubleAttr_Obj, 0);
            c[i][k].set(GRB_StringAttr_VarName, vname.str());
        }
    }

    x = new GRBVar** [njob];
    for(int i= 0; i<njob; i++)
    {

```

```

x[i] = new GRBVar* [njob];
for(int j= 0; j<njob; j++)
{
    x[i][j] = model.addVars(nmaq, GRB_BINARY);
    model.update();
    for (int k= 0; k< nmaq; k++)
    {
        ostringstream vname;
        vname << "x" << i << j << k;
        x[i][j][k].set(GRB_DoubleAttr_Obj, 0);
        x[i][j][k].set(GRB_StringAttr_VarName, vname.str());
    }
}

//-----
//função objetivo: min sum from {i=1} to n C_{i m}
//-----
GRBLinExpr obj = 0;//cria a expressao da função objetivo
for(int i= 0; i<njob; i++)//somatorio em i
    obj+=c[i][job[i].maq[nmaq-1]];
model.setObjective(obj, GRB_MINIMIZE);
model.update(); //atualiza o modelo

//-----
//restrição 1 : C_{i, maq_1} geslant p_{i, maq_1} forall i=1,...,n
//-----
for (int i= 0; i< njob; i++)//para todo i
    model.addConstr(c[i][job[i].maq[0]]>=job[i].tp[0], "r1"); //
adicionando restrição 1

//-----
//restrição 2 : C_{i, maq_{k+1}} geslant C_{i, maq_k} + p_{i,
maq_{k+1}} forall i=1,...,n forall k=1,...,m-1
//-----
for (int i= 0; i< njob; i++)//para todo i
    for (int k= 0; k< nmaq-1; k++)//para todo k

        model.addConstr(c[i][job[i].maq[k+1]]>=c[i][job[i].maq[k]]+job[i].tp[k+1], "r2"); //
adicionando restrição 2

//-----
//restrição 3 : C_{ijk} geslant C_{ik} + p_{jk} - M (1-x_{ijk}) i<>j forall
i,j=1,...,n forall k=1,...,m
//-----
for (int i= 0; i< njob; i++)//para todo i
    for (int j= 0; j< njob; j++)//para todo j
        if(i!=j)//i diferente de j
            for (int k= 0; k< nmaq; k++)//para todo k

```

```

                                model.addConstr(c[j][k]>=c[i][k]+job[j].tp2[k]-
M*(1-x[i][j][k]), "r3"); //adicionando restri  o 3*/

//-----
//restri  o 4 : C_{ik} geslant C_{jk} + p_{ik} - M x_{ijk} i<>j forall
i,j=1,...,n forall k=1,...,m
//-----
for (int i= 0; i< njob; i++)//para todo i
    for (int j= 0; j< njob; j++)//para todo j
        if(i!=j)//i diferente de j
            for (int k= 0; k< nmaq; k++)//para todo k
                model.addConstr(c[i][k]>=c[j][k]+job[i].tp2[k]-
M*x[i][j][k], "r4"); //adicionando restri  o 4*/

//-----
//restri  o 5 : C_{ik} geslant 0 forall i=1,...,n forall k=1,...,m
//-----
for (int i= 0; i< njob; i++)//para todo i
    for (int k= 0; k< nmaq; k++)//para todo k
        model.addConstr(c[i][k]>=0, "r5"); // adicionando
restri  o 5

//-----
//inicia a resolu  o do modelo
//-----
model.update();
model.write("gurobi109.modelo.lp");
model.optimize();

//-----
//Exporta a solu  o
//-----
FO= model.get(GRB_DoubleAttr_ObjVal);
for (int i= 0; i< njob; i++)
    for (int k= 0; k< nmaq; k++)
        respostaC[i][k]= c[i][k].get(GRB_DoubleAttr_X);
for (int i= 0; i< njob; i++)
    for (int j= 0; j< njob; j++)
        for (int k= 0; k< nmaq; k++)
            respostaX[i][j][k]= x[i][j][k].get(GRB_DoubleAttr_X);

}catch(GRBException e)
{
    cout << "Gurobi - C  digo do erro = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}catch(...)
{
    cout << "Gurobi - Erro durante otimiz  o" << endl;
}

```

```

delete[] c;
delete[] x;
delete env;//termina ambiente gurobi*/

time(&t_fim);
tempo= difftime(t_fim,t_ini);
cout<<"O problema foi resolvido em "<<tempo<<" segundos.\n";
cout<<"\n-----\n\n";
}

//-----
//Função heurística construtiva: heurística construtiva para o problema de Job
Shop
//-----
void heuristicaconstrutiva()
{
    cout<<"\nIniciando a heurística construtiva.\n\n";
    time(&t_ini);

    //Backup dos dados de entrada
    jobs dado;
    vector <jobs> job2;
    job2.resize(0);
    for (int i=0; i<njob; i++)
    {
        for (int k=0; k<nmaq; k++)
        {
            dado.maq[k]= job[i].maq[k];
            dado.tp[k]= job[i].tp[k];
            dado.tp2[k]= job[i].tp2[k];
        }
        job2.push_back(dado);
    }

    int menor, posmenortempo, postarefamaqanterior;

    for (int k=0; k<nmaq; k++) //para todas as máquinas
        for (int j=0; j<njob; j++) //para todas as tarefas
        {
            menor= M; //o menor tempo recebe M
            for (int i=0; i<njob; i++)
                if (job2[i].tp[k]< menor) //procura o menor tempo
                {
                    menor= job2[i].tp[k]; //atualiza o valor do menor
                    posmenortempo= i; //guarda a posição do menor
                }
        }
}

```

[illegible]

```

        maquina[job2[posmenortempo].maq[k]].tempoini[i]=
maquina[job2[posmenortempo].maq[k]].tempofim[i-1];
    }

    maquina[job2[posmenortempo].maq[k]].tempofim[i]=
maquina[job2[posmenortempo].maq[k]].tempoini[i] +job2[posmenortempo].tp[k];
    //if (na propria maquina< na outra maquina)
    }
    break;
}
    job2[posmenortempo].tp[k]= M; //atualiza o valor do menor
tempo, para não ser selecionado novamente
}

calculafo(0); //Cálculo da FO da solução s

cout<<"\nHeurística construtiva concluída.\n\n";
time(&t_fim);
tempo= difftime(t_fim,t_ini);
cout<<"FO = "<<FO<<endl;
cout<<"O problema foi resolvido em "<<tempo<<" segundos.\n";
}

//-----
//Função calculafo: Calcula o valor da FO da solução s
//-----
void calculafo(int s)
{
    switch (s)
    {
        case 0:
            FO= 0;
            for (int i=0; i<njob; i++) //para todas as tarefas
                for (int j=0; j<njob; j++) //para todas as tarefas
                    if(maquina[job[i].maq[nmaq-1]].tarefa[j]== i)
                        FO+= maquina[job[i].maq[nmaq-
1]].tempofim[j];
            break;

        case 1:
            FOs1= 0;
            for (int i=0; i<njob; i++) //para todas as tarefas
                for (int j=0; j<njob; j++) //para todas as tarefas
                    if(maquina1[job[i].maq[nmaq-1]].tarefa[j]== i)
                        FOs1+= maquina1[job[i].maq[nmaq-
1]].tempofim[j];
            break;

        case 2:

```

```

        FOs2= 0;
        for (int i=0; i<njob; i++)          //para todas as tarefas
            for (int j=0; j<njob; j++)      //para todas as tarefas
                if(maquina2[job[i].maq[nmaq-1]].tarefa[j]== i)
                    FOs2+= maquina2[job[i].maq[nmaq-
1]].tempofim[j];
            break;
        }
    }

//-----
//Função calculatempo: Calcula os tempos iniciais e finais de cada tarefa da
solução
//-----
void calculatempo(int s)
{
    int seqtar[njob], seqmaq[nmaq];
    int tempofimtar[njob], tempofimmaq[nmaq];
    int inviabilidade=0, totaltaratribuidas= 0;

    switch (s)
    {
        case 0:
            for (int i=0; i<njob; i++)      //para todas as tarefas
                for (int k=0; k<nmaq; k++)  //para todas as
máquinas
                    {
                        maquina[k].tempoini[i]= 0;
                        maquina[k].tempofim[i]= 0;
                    }
            for (int i=0; i<njob; i++)      //para todas as tarefas
            {
                seqtar[i]= 0;
                tempofimtar[i]= 0;
            }
            for (int k=0; k<nmaq; k++)      //para todas as máquinas
            {
                seqmaq[k]= 0;
                tempofimmaq[k]= 0;
            }
            do{
                for (int i=0; i<njob; i++)  //para todas as tarefas
                    if(seqtar[i]<nmaq)      //se a sequencia
da tarefa for menor que o total de máquinas

                        if(maquina[job[i].maq[seqtar[i]]].tarefa[seqmaq[job[i].maq[seqtar[i]]]]==i)
                            {

```

```

        if((seqmaq[job[i].maq[seqtar[i]]]==0)&&(seqtar[i]==0)) //Ã© primeira tarefa da
        mÃquina e Ã© a primeira mÃquina da tarefa
            {

                maquina[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]]= 0;

                maquina[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]]=
                job[i].tp[seqtar[i]];

                tempofimmaq[job[i].maq[seqtar[i]]]=
                maquina[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];
                tempofimtar[i]=
                maquina[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];

                seqmaq[job[i].maq[seqtar[i]]]++; //sequencia de mÃquinas alocadas
                seqtar[i]++;

                //sequencia de tarefas alocadas
            }else
            {

                if(seqmaq[job[i].maq[seqtar[i]]]==0) //Ã© a primeira tarefa da mÃquina mas
                nÃo Ã© a primeira mÃquina da tarefa

                    maquina[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]]=
                    tempofimtar[i];

                    else if(seqtar[i]==0) //nÃo Ã© a
                    primeira tarefa da mÃquina mas Ã© a primeira mÃquina da tarefa

                        maquina[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]]=
                        tempofimmaq[job[i].maq[seqtar[i]]];

                        else //nÃo Ã© a primeira tarefa
                        da mÃquina nem Ã© a primeira mÃquina da tarefa

                            if (tempofimtar[i] >
                            maquina[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]-1]) //escolha o
                            maior tempo

                                maquina[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]]=
                                tempofimtar[i];

                                else

                                    maquina[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]]=
                                    maquina[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]-1];

                                    maquina[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]]=
                                    maquina[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]+job[i].tp[seqtar[i]]
                                    ];

                                    tempofimmaq[job[i].maq[seqtar[i]]]=
                                    maquina[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];

```

```

tempofimtar[i]=
maquina[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];

seqmaq[job[i].maq[seqtar[i]]]++; //sequencia de máquinas alocadas
seqtar[i]++;
//sequencia de tarefas alocadas
}
totaltaratribuidas++;
inviabilidade= 0;
} else //soma o total de inviabilidades
{
inviabilidade++;
if(inviabilidade>njob)
{
totaltaratribuidas= (njob*nmaq);
for (int i=0; i<njob; i++)

//para todas as tarefas
for (int k=0; k<nmaq;
k++) //para todas as máquinas
{

maquina[k].tempoini[i]= 999999;

maquina[k].tempofim[i]= 999999;

}

}

} while(totaltaratribuidas<(njob*nmaq)); //enquanto houver
tarefas não percorridas
break;

case 1:
for (int i=0; i<njob; i++) //para todas as tarefas
for (int k=0; k<nmaq; k++) //para todas as
máquinas
{
maquina1[k].tempoini[i]= 0;
maquina1[k].tempofim[i]= 0;
}
for (int i=0; i<njob; i++) //para todas as tarefas
{
seqtar[i]= 0;
tempofimtar[i]= 0;
}
for (int k=0; k<nmaq; k++) //para todas as máquinas
{
seqmaq[k]= 0;
tempofimmaq[k]= 0;
}

```

```

do{
    for (int i=0; i<njob; i++)          //para todas as tarefas
(vector job)
        if(seqtar[i]<nmaq)              //se a sequencia
da tarefa for menor que o total de máquinas

            if(maquina1[job[i].maq[seqtar[i]]].tarefa[seqmaq[job[i].maq[seqtar[i]]]]==i)
            {

                if((seqmaq[job[i].maq[seqtar[i]]]==0)&&(seqtar[i]==0)) //a primeira tarefa da
máquina e a primeira máquina da tarefa
                {

                    maquina1[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]] = 0;

                    maquina1[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]] =
job[i].tp[seqtar[i]];

                    tempofimmaq[job[i].maq[seqtar[i]]] =
maquina1[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];
                    tempofimtar[i] =
maquina1[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];

                    seqmaq[job[i].maq[seqtar[i]]]++; //sequencia de máquinas alocadas
                    seqtar[i]++;
                //sequencia de tarefas alocadas
                }else
                {

                    if(seqmaq[job[i].maq[seqtar[i]]]==0) //a primeira tarefa da máquina mas
não a primeira máquina da tarefa

                        maquina1[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]] =
tempofimtar[i];

                        else if(seqtar[i]==0) //não a primeira tarefa da máquina mas a primeira máquina da tarefa

                            maquina1[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]] =
tempofimmaq[job[i].maq[seqtar[i]]];

                            else //não a primeira tarefa
da máquina nem a primeira máquina da tarefa

                                if (tempofimtar[i] >
maquina1[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]-1]) //escolha o
maior tempo

                                    maquina1[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]] =
tempofimtar[i];

                                    else

```

```

    maquina1[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]] =
maquina1[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]-1];

    maquina1[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]] =
maquina1[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]] + job[i].tp[seqtar[
i]];

    tempofimmaq[job[i].maq[seqtar[i]]] =
maquina1[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];
    tempofimtar[i] =
maquina1[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];

    seqmaq[job[i].maq[seqtar[i]]]++; //sequencia de máquinas alocadas
    seqtar[i]++;

    //sequencia de tarefas alocadas
    }
    totaltaratribuidas++;
    inviabilidade = 0;
} else //soma o total de inviabilidades
{
    inviabilidade++;
    if(inviabilidade > njob)
    {
        totaltaratribuidas = (njob * nmaq);
        for (int i = 0; i < njob; i++)

//para todas as tarefas
        for (int k = 0; k < nmaq;
k++) //para todas as máquinas
        {

            maquina1[k].tempoini[i] = 999999;

            maquina1[k].tempofim[i] = 999999;

        }
    }
}

} while(totaltaratribuidas < (njob * nmaq)); //enquanto houver
tarefas não percorridas
break;

case 2:
    for (int i = 0; i < njob; i++) //para todas as tarefas
        for (int k = 0; k < nmaq; k++) //para todas as
máquinas
        {
            maquina2[k].tempoini[i] = 0;
            maquina2[k].tempofim[i] = 0;
        }
    for (int i = 0; i < njob; i++) //para todas as tarefas

```

```

{
    seqtar[i]= 0;
    tempofimtar[i]= 0;
}
for (int k=0; k<nmaq; k++)          //para todas as máquinas
{
    seqmaq[k]= 0;
    tempofimmaq[k]= 0;
}

do{
    for (int i=0; i<njob; i++)          //para todas as tarefas
(vector job)
        if(seqtar[i]<nmaq)              //se a sequencia
da tarefa for menor que o total de máquinas

            if(maquina2[job[i].maq[seqtar[i]]].tarefa[seqmaq[job[i].maq[seqtar[i]]]]==i)
                {

                    if((seqmaq[job[i].maq[seqtar[i]]]==0)&&(seqtar[i]==0)) //a primeira tarefa da
máquina e a primeira máquina da tarefa
                        {

                            maquina2[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]] = 0;

                            maquina2[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]] =
job[i].tp[seqtar[i]];

                            tempofimmaq[job[i].maq[seqtar[i]]] =
maquina2[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];
                            tempofimtar[i] =
maquina2[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]];

                            seqmaq[job[i].maq[seqtar[i]]]++; //sequencia de máquinas alocadas
seqtar[i]++;
                        //sequencia de tarefas alocadas
                        }else
                        {

                            if(seqmaq[job[i].maq[seqtar[i]]]==0) //a primeira tarefa da máquina mas
não a primeira máquina da tarefa

                                maquina2[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]] =
tempofimtar[i];

                                else if(seqtar[i]==0) //não a primeira tarefa da máquina mas a primeira máquina da tarefa

                                    maquina2[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]] =
tempofimmaq[job[i].maq[seqtar[i]]];

```

```

else // não é a primeira tarefa
da máquina nem é a primeira máquina da tarefa
    if (tempofimtar[i] >
maquina2[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]-1]) //escolha o
maior tempo

    maquina2[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]]=
tempofimtar[i];

    else

    maquina2[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]]=
maquina2[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]-1];

    maquina2[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]]]=
maquina2[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]+job[i].tp[seqtar[
i]];

    tempofimmaq[job[i].maq[seqtar[i]]]=
maquina2[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]];
    tempofimtar[i]=
maquina2[job[i].maq[seqtar[i]]].tempofim[seqmaq[job[i].maq[seqtar[i]]];

    seqmaq[job[i].maq[seqtar[i]]]++; //sequencia de máquinas alocadas
    seqtar[i]++;

    //sequencia de tarefas alocadas
    }
    totaltaratribuidas++;
    inviabilidade= 0;
} else //soma o total de inviabilidades
{
    inviabilidade++;
    if(inviabilidade>njob)
    {
        totaltaratribuidas= (njob*nmaq);
        for (int i=0; i<njob; i++)

        //para todas as tarefas
        for (int k=0; k<nmaq;
k++) //para todas as máquinas
        {

            maquina2[k].tempoini[i]= 999999;

            maquina2[k].tempofim[i]= 999999;

        }
    }
}

} while(totaltaratribuidas<(njob*nmaq)); //enquanto houver
tarefas não percorridas
break;
}

```

```

}

//-----
//Função troca1opt: faz uma troca 1 optimal na solução s
//-----
void troca1opt(int i, int j, int maq, int s)
{
    int troca, aux;

    switch (s)
    {
        case 0:
            //troca na máquina maq, a tarefa i por tarefa j
            troca= maquina[maq].tarefa[i];
            maquina[maq].tarefa[i]= maquina[maq].tarefa[j];
            maquina[maq].tarefa[j]= troca;
            break;

        case 1:
            //troca na máquina maq, a tarefa i por tarefa j
            troca= maquina1[maq].tarefa[i];
            maquina1[maq].tarefa[i]= maquina1[maq].tarefa[j];
            maquina1[maq].tarefa[j]= troca;
            break;

        case 2:
            //troca na máquina maq, a tarefa i por tarefa j
            troca= maquina2[maq].tarefa[i];
            maquina2[maq].tarefa[i]= maquina2[maq].tarefa[j];
            maquina2[maq].tarefa[j]= troca;
            break;
    }
}

//-----
//Função troca2opt: faz uma troca 2 optimal na solução s
//-----
void troca2opt(int s)
{
    int i, j, maq;

    //Escolhe aleatoriamente as tarefas a serem trocas
    for(int l=0; l<2; l++)
    {
        i= rand()%(njob-1);
        j= rand()%(njob-1);
        maq= rand()%(nmaq-1);
        troca1opt(i, j, maq, s);
    }
}

```

```

//-----
//Função troca3opt: faz uma troca 3 optimal na solução s
//-----
void troca3opt(int s)
{
    int i, j, maq;

    //Escolhe aleatoriamente as tarefas a serem trocas
    for(int l=0; l<3; l++)
    {
        i= rand()%(njob-1);
        j= rand()%(njob-1);
        maq= rand()%(nmaq-1);
        troca1opt(i, j, maq, s);
    }
}

//-----
//Função troca4opt: faz uma troca 4 optimal na solução s
//-----
void troca4opt(int s)
{
    int i, j, maq;

    //Escolhe aleatoriamente as tarefas a serem trocas
    for(int l=0; l<4; l++)
    {
        i= rand()%(njob-1);
        j= rand()%(njob-1);
        maq= rand()%(nmaq-1);
        troca1opt(i, j, maq, s);
    }
}

//-----
//Função troca5opt: faz uma troca 5 optimal na solução s
//-----
void troca5opt(int s)
{
    int i, j, maq;

    //Escolhe aleatoriamente as tarefas a serem trocas
    for(int l=0; l<5; l++)
    {
        i= rand()%(njob-1);
        j= rand()%(njob-1);
        maq= rand()%(nmaq-1);
        troca1opt(i, j, maq, s);
    }
}

```



```

        maquina3.resize(0);          //solução s3 recebe
solução s2
        for (int l=0; l<nmaq; l++)
            maquina3.push_back(maquina2[l]);
        FOs3= FOs2;
        maquina2.resize(0);          //solução s2 recebe
solução s1
        for (int l=0; l<nmaq; l++)
            maquina2.push_back(maquina1[l]);
        FOs2= FOs1;
    }
}
if(FOs3<FOs1)
{
    maquina2.resize(0);          //solução s2 recebe solução s3
    for (int l=0; l<nmaq; l++)
        maquina2.push_back(maquina3[l]);
    FOs2= FOs3;
}else{
    maquina2.resize(0);          //solução s2 recebe solução s1
    for (int l=0; l<nmaq; l++)
        maquina2.push_back(maquina1[l]);
    FOs2= FOs1;
}
}

//-----
//Função formatacao: formata a saída do gurobi para poder ser usada no VNS
//-----
void formatacao()
{
    int contador= 0, posicao= 0, ordem[MAX_JOB];

    for (int k=0; k<nmaq; k++)
    {
        for (int i=0; i<njob; i++)
        {
            contador= 0;
            for (int j=0; j<njob; j++)
                contador+= respostaX[i][j][k];
            posicao= njob-contador-1;
            ordem[posicao]= i;
        }
        for (int i=0; i<njob; i++)
            maquina[k].tarefa[i]= ordem[i];
    }
    calculatempo(0);
}
//-----
//Função vns: Controla a execução da busca em vizinhança variável

```

```
//-----
void vns()
{
    cout<<"\nIniciando a metaheurística VNS.\n\n";
    time(&t_ini);

    int i, j, maq;
    int iter= 0;           //número atual de iterações
    int k= 0;              //estrutura de vizinhança atual

    maquina0.resize(0);    //solução s0 recebe solução s
    for (int l=0; l<nmaq; l++)
        maquina0.push_back(maquina[l]);
    FOs0= FO;
    maquina1.resize(0);    //solução s1 recebe solução s
    for (int l=0; l<nmaq; l++)
        maquina1.push_back(maquina[l]);
    FOs1= FO;

    while (iter<ITERMAX)   //enquanto critério de parada não for
    atendido
    {
        k= 1;              //estrutura de vizinhança atual
        iter++;            //conta o número de iterações
        cout<<"\nIteração "<<iter<<" - FO "<<FO;
        while (k<= KMAX)
        {
            switch (k)
            {
                case 1:
                    i= rand()%(njob-1);
                    j= rand()%(njob-1);
                    maq= rand()%(nmaq-1);
                    troca1opt(i, j, maq, 1);
                    calculatempo(1);
                    calculafo(1);
                    break;

                case 2:
                    troca2opt(1);
                    calculatempo(1);
                    calculafo(1);
                    break;

                case 3:
                    troca3opt(1);
                    calculatempo(1);
                    calculafo(1);
                    break;
            }
        }
    }
}
```

```

        case 4:
            troca4opt(1);
            calculatempo(1);
            calculafo(1);
            break;

        case 5:
            troca5opt(1);
            calculatempo(1);
            calculafo(1);
            break;

        case 6:
            troca6opt(1);
            calculatempo(1);
            calculafo(1);
            break;
    }
    buscalocal();
    if (FOs2<FO)
    {
        soluÃ§Ã£o s2
        soluÃ§Ã£o s2
        maquina.resize(0);          //soluÃ§Ã£o s recebe
        maquina1.resize(0);         //soluÃ§Ã£o s1 recebe

        for (int l=0; l<nmaq; l++)
        {
            maquina.push_back(maquina2[l]);
            maquina1.push_back(maquina2[l]);
        }
        FO= FOs2;
        FOs1= FOs2;
        k= 1;
        iter= 0; //zera o nÃºmero de iteraÃ§Ãµes sem melhora
    }else
    {
        k++;
        soluÃ§Ã£o s
        maquina1.resize(0);          //soluÃ§Ã£o s1 recebe

        for (int l=0; l<nmaq; l++)
            maquina1.push_back(maquina[l]);
        FOs1= FO;
    }
}

if(FOs0<FO)
{
    maquina.resize(0);          //soluÃ§Ã£o s recebe soluÃ§Ã£o s0
    for (int l=0; l<nmaq; l++)

```

```

        maquina.push_back(maquina0[l]);
        FO= FOs0;
    }

    cout<<"\nMetaheurística VNS concluída.\n\n";
    time(&t_fim);
    tempo= difftime(t_fim,t_ini);
    cout<<"O problema foi resolvido em "<<tempo<<" segundos.\n";
}

//-----
//Função imprime: imprime os resultados no arquivo gurobi109.saida.txt
//-----
void imprime(int s)
{
    int contador= 0, posicao= 0, ordem[MAX_JOB];

    ofstream destino ("gurobi109.saida.txt");
    if (!destino)
        cerr << "\nErro ao abrir o arquivo gurobi109.saida.txt\n\n";
    destino<<"\n\tFO: "<<FO<< endl;

    destino<< njob<<" "<<nmaq<< endl;;
    for (int i=0; i<njob; i++)
    {
        for (int k=0; k<nmaq; k++)
        {
            destino.width(2);
            destino<< job[i].maq[k]<<" ";
            destino.width(2);
            destino<< job[i].tp[k]<<" ";
        }
        destino<< endl;
    }

    switch (s)
    {
        case 0: //solução do GUROBI
            destino<< "\n\n\tC[i,k]:\n\n";
            for (int i=0; i<njob; i++)
            {
                for (int k=0; k<nmaq; k++)
                {
                    destino.width(4);
                    destino<< respostaC[i][k]<<" ";
                }
                destino<< endl;
            }

            destino<< "\n\n\tX[i,j,k]:\n\n";

```

```

for (int k=0; k<nmaq; k++)
{
    destino<< "Máquina " << k << ": \n";
    for (int i=0; i<njob; i++)
    {
        contador= 0;
        for (int j=0; j<njob; j++)
        {
            destino.width(2);
            destino<< respostaX[i][j][k] << " ";
            contador+= respostaX[i][j][k];

        }
        posicao= njob-contador-1;
        ordem[posicao]= i;
        destino<< endl;
    }
    destino<< "\nSequência de produção na máquina
" << k << ": \nTarefa";

    for (int i=0; i<njob; i++)
    {
        maquina[k].tarefa[i]= ordem[i];
        destino<< " -> " << ordem[i];
    }
    destino<< "\n\n";
}
calculatempo(0);

destino<< "\n\n\tVector máquinas: \n\n";
for (int k=0; k<nmaq; k++)
{
    for (int i=0; i<njob; i++)
    {
        destino.width(2);
        destino<< maquina[k].tarefa[i] << " ( ";
        destino.width(2);
        destino<< maquina[k].tempoini[i] << " , ";
        destino.width(2);
        destino<< maquina[k].tempofim[i] << " ) ";
    }
    destino<< endl;
}
break;

case 1: //solução da heurística e da metaheurística
destino<< "\n\n\tVector máquinas: \n\n";
for (int k=0; k<nmaq; k++)
{
    for (int i=0; i<njob; i++)
    {

```

```

                                destino.width(2);
                                destino<< maquina[k].tarefa[i]<<" ( ";
                                destino.width(2);
                                destino<< maquina[k].tempoini[i]<<" , ";
                                destino.width(2);
                                destino<< maquina[k].tempofim[i]<<" ) ";
                                }
                                destino<< endl;
                                }
                                break;
                                }
                                cout<<"\nArquivo gurobi109.saida.txt impresso.\n";
                                }

```