UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ADMINISTRAÇÃO, CIÊNCIAS CONTÁBEIS,
ENGENHARIA DE PRODUÇÃO E SERVIÇO SOCIAL

ADRIEL HENRIQUE BEATO DE FRANÇA

UM ESTUDO SOBRE A APLICAÇÃO DE HEURÍSTICAS NO PROBLEMA DE JOB SHOP

ITUIUTABA
2023

ADRIEL HENRIQUE BEATO DE FRANÇA

UM ESTUDO SOBRE A APLICAÇÃO DE HEURÍSTICAS NO PROBLEMA DE JOB SHOP

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
2023

UM ESTUDO SOBRE A APLICAÇÃO DE HEURÍSTICAS NO PROBLEMA DE JOB SHOP

Trabalho de Conclusão de Curso para a conclusão da engenharia de produção na Universidade Federal de Uberlândia pela banca examinadora formada por:

Ituiutaba, dia do mês do ano.
Banca Examinadora:

Jorge von Atzingen dos Reis (orientador)
Universidade Federal de Uberlândia

Eugênio Pacceli Costa (membro da banca)
Universidade Federal de Uberlândia

Vanessa Aparecida de Oliveira Rosa (membro da banca)
Universidade Federal de Uberlândia

Dedico este trabalho à Melissa.
A Deus por ter me dado
força durante toda a graduação.
Aos meus familiares que me forneceram suporte
e sempre me incentivaram a estudar.
Aos meus amigos e parceiros de graduação.

AGRADECIMENTOS

Agradeço à Universidade Federal de Uberlândia como instituição.

Também agradeço a todos os professores da instituição que durante todo o meu período de graduação me transmitiram não somente conteúdos teóricos como também uma carga de fibra moral que eu vou levar para a vida e com toda a certeza vai me ajudar a impactar o mundo de maneira positiva e não mediram esforços em investir tempo no crescimento de quem realmente quer ser grande.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPQ/UFU) por financiar este projeto desenvolvido.

Agradeço ao meu orientador, o professor Dr. Jorge von Atzingen dos Reis por me dar suporte e apoio durante todo o período de projeto e também por me mostrar que não existem caminhos certos ou errados, existem escolhas.

Agradeço à banca que com seus comentários e avaliações enriquecerá a qualidade do trabalho.

O artista que não duvida consegue pouco.

- Leonardo da Vinci

RESUMO

O Problema de *Job Shop* é classificado como um problema *NP-hard*, ou seja, não existe um algoritmo que encontre a solução ideal em tempo polinomial. Dessa forma a resolução do problema de sequenciamento de tarefas tem fundamental importância na gestão estratégica que afeta diretamente todas as diretrizes de uma companhia, além de se enquadrar nos problemas clássicos dentro do planejamento e controle da produção. O presente trabalho aborda um estudo onde é necessário desenvolver e implementar um modelo matemático capaz de determinar o sequenciamento ótimo da produção, o qual retornará a solução ótima referente a melhor combinação e proporção de produção e fornecimento de determinados produtos ou serviços, porém demanda um tempo elevado para retornar tal solução. O estudo é baseado na formulação do modelo matemático, a utilização de técnicas de otimização para resolução do modelo e análise dos resultados obtidos, com intuito de buscar a maximização de lucros da operação logística, ou seja, buscar o preenchimento de certas lacunas no mercado. O presente trabalho consiste no comparativo de uma nova heurística construtiva e heurística de refinamento que vise a melhoria dos resultados obtidos pelas meta-heurísticas ILS e VNS para problemas de *Job Shop* implementados na linguagem de programação C++. Esta pesquisa teve início com a resolução do problema através da programação linear com o objetivo deste resultado servir de parâmetro para a qualidade da resposta obtida com as meta-heurísticas ILS e VNS. Posteriormente as meta-heurísticas ILS e VNS foram implementadas utilizando a heurística construtiva de Baker e Trietsch e esses resultados foram analisados junto com as respostas obtidas por programação linear. Foi possível concluir que os *jobs* com maior tempo total tinham prioridade de execução, resultando na proposição de um meio diferente de montar a heurística construtiva e para determinadas instâncias de maiores dimensões do problema utilizar uma heurística de refinamento. Esse método tem o objetivo de atingir uma resposta otimizada em um tempo menor de execução, de modo a auxiliar na tomada de decisão em um sequenciamento de trabalhos em máquinas.

Palavras-chave: *Job Shop*, Programação Linear, Otimização Combinatória, Heurística

ABSTRACT

The Job Shop Problem is classified as an NP-hard problem, there is no algorithm that finds the ideal solution in polynomial time. The resolution of the task sequencing problem is fundamental in strategic management that directly affects all the guidelines of a company, in addition to fitting into the classic problems within production planning and control. The present work study where it is necessary to develop and implement a mathematical model capable of determining the optimal sequencing of production, which will return the optimal solution referring to the best combination and proportion of production and supply of certain products or services, but it takes time high to return such a solution. The study is based on the formulation of the mathematical model, the use of optimization techniques for solving the model and analysis of the results obtained, with the aim of seeking to maximize profits from the logistics operation, that is, to seek to fill certain gaps in the market. The present work compares a new constructive heuristic and a refinement heuristic that aims to improve the results obtained by the ILS and VNS meta-heuristics for Job Shop problems implemented in the C++ programming language. This research began with the resolution of the problem through linear programming with the objective of this result serving as a parameter for the quality of the response obtained with the meta-heuristics ILS and VNS. Subsequently, the ILS and VNS meta-heuristics were implemented using Baker and Trietsch's constructive heuristic and these results were analyzed together with the responses obtained by linear programming. It was possible to conclude that jobs with greater total time had execution priority, resulting in the proposition of a different way of setting up the constructive heuristic and for certain instances of larger dimensions of the problem to use a refinement heuristic. This method aims to achieve an optimized response in a shorter execution time, in order to assist in decision making in sequencing work on machines.

Keywords: Job Shop, Linear Programming, Combinatorial Optimization, Heurístic

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

FO	Função Objetivo
ILS	*Iterated Local Search*
JSP	*Job Shop Problem*
PCP	Planejamento e Controle da Produção
PO	Pesquisa Operacional
VNS	*Variable Neighborhood Search*

SUMÁRIO

1 INTRODUÇÃO

Nesta seção serão abordados os objetivos da pesquisa em questão, objetivos gerais e específicos, procedimentos metodológicos e sua delimitação.

1.1 Contextualização e justificativa

Existem diversos tipos de problemas que são classificados como problemas de planejamento e programação da produção. Um dos métodos para determinar uma possível solução é por meio da programação linear. 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 uma parte importante do Planejamento e Controle da Produção, devido ao fato que um plano eficiente aumenta a produtividade mesmo utilizando recursos restritos, como máquina e rotinas de trabalho. (ARENALES, 2007).

Para atingir esse objetivo, segundo Caetano (2008) a busca do melhoramento contínuo das operações dentro da organização é possível utilizando técnicas de Pesquisa Operacional (PO) e estudando os seus métodos. Na PO existem três principais categorias de métodos para a resolução de problemas: os métodos determinísticos, os métodos estocásticos e os métodos aproximados. Os métodos determinísticos utilizam programação linear, programação inteira, programação dinâmica e otimização em redes. São técnicas nas quais é necessária uma modelagem matemática simplificada que resulta em uma solução ótima. Nos métodos aproximados utilizam as heurísticas e as meta-heurísticas, ou seja, é construído um algoritmo e utiliza-se regras operacionais para encontrar a melhor solução possível em determinado espaço de tempo, sendo as soluções boas ou melhoradas. Os métodos estocásticos utilizam a teoria das filas, teoria dos jogos e simulação de eventos discretos. Existe uma distribuição de probabilidade dos valores que varia com o tempo, tornando as variáveis em parte aleatórias (TAHA, 2008).

1.2 Objetivos da pesquisa

Esta pesquisa possui como objetivo principal explorar diferentes heurísticas para encontrar uma boa solução para o *Job Shop Problem* com o intuito de diminuir o tempo de solução.

1.2.1 Objetivos Específicos

Implementar o modelo matemático do *Job Shop Problem*, meta-heurística *Variable Neighborhood Search*, heurística construtiva, heurística de refinamento, meta-heurística *Iterated Local Search* para a resolução do *Job Shop Problem* e encontrar a melhor forma de resolver esse problema comparando os resultados de todos os métodos utilizados.

1.3 Procedimentos metodológicos

De início foi realizado um entendimento do *Job Shop Problem* por meio de uma revisão da literatura, vídeos e artigos. Em seguida foi feita a busca de dados no *OR-Library* o qual apresenta dados confiáveis para estudo, teste e comparações referentes a problemas clássicos. Após a busca, foi implementado o modelo matemático, heurística construtiva, heurística de refinamento, meta-heurística *Iterated Local Search*, meta-heurística *Variable Neighborhood Search*. Posteriormente foi feita a comparação de resultados de todos os métodos.

1.4 Relevância da Pesquisa

A realização da ordenação das tarefas é de total importância para diminuir custos. Dessa forma, calculando a melhor forma de ordenar as tarefas, consequentemente há um aumento no lucro e na competitividade da empresa em seu mercado atual. Entretanto, realizar esse método manualmente pode demorar principalmente se o número de máquinas e tarefas for grande e dificilmente chegaria a uma solução ótima. Todavia, aplicando métodos computacionais a solução é de melhor qualidade e a mesma técnica pode ser aplicada em diversos bancos de dados, sem precisar de alterações no código em si.

1.5 Delimitação do trabalho

No presente trabalho foram utilizados dados teóricos para os testes considerando como solução ótima ordenação de menor *makespan,* que é o tempo total necessário para completar um conjunto de tarefas. Entretanto, desconsidera as possíveis variáveis humanas que podem interferir no resultado de todo o processo e também variações nos processos executados pelas máquinas ao longo do tempo, sendo de pequena magnitude ou não, característica inerente a qualquer processo produtivo.

1.6 Estrutura do trabalho

Na primeira seção do trabalho é abordada uma fundamentação teórica com os temas relacionados à pesquisa: Planejamento e Controle da Produção e *Job Shop Problem* (JSP) e métodos para resolver o mesmo. Após a fundamentação teórica foram apresentados, de forma detalhada, os métodos dentro da PO escolhidos para resolver o Problema de *Job Shop Problem* (JSP). Os escolhidos foram: Heurística de Baker e Trietsch combinada com *Iterated Local Search* (ILS), Heurística de Baker e Trietsch combinada com *Variable Neighborhood Search* (VNS), Heurística construtiva combinada com VNS, Heurística Construtiva combinada com ILS, Heurística construtiva combinada com Heurística de refinamento e VNS, Heurística Construtiva combinada com Heurística de refinamento e ILS.

Ademais, foram esquematizadas as estruturas de vizinhança, heurística e busca local a fim de mostrar a lógica por trás de cada uma das estruturas. Posteriormente foram apresentados os resultados dos dados da OR-Library em todos os métodos de resolução, posteriormente, nas considerações finais são abordados os melhores métodos, conclusão do trabalho, delimitações do trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nessa seção serão abordados os temas relacionados ao Planejamento e Controle da Produção. Além disso, será realizado um detalhamento do *Job Shop Problem* (JSP) e métodos para resolução desse problema clássico da literatura, incluindo as heurísticas desenvolvidas.

2.1 Planejamento e Controle da Produção (PCP)

O planejamento e o controle se preocupam em gerenciar as atividades da operação produtiva da empresa de maneira que a demanda do consumidor final seja satisfeita, coordenando e apoiando todo o sistema. Esse sistema se caracteriza pela transformação das entradas em saídas e se envolve direta ou indiretamente com a produção seja para buscar informações ou recursos para que todos o planejamento seja executado. Os planos de controle são essenciais em qualquer operação produtiva e quanto maior é o contato com o consumidor final mais difícil se torna criar esse plano tendo em vista a natureza imediata de suas operações (VEGGIAN, 2011).

De acordo com Tubino (2007) o acompanhamento e controle da produção tem a finalidade de concretizar a ligação entre o planejamento e a execução das atividades operacionais, visa reconhecer os desvios juntamente a suas magnitudes, oferecendo meios para que os responsáveis pelas ações corretivas possam agir. Quanto maior o nível de eficiência as ações de acompanhamento atingirem, menores serão os desvios a serem corrigidos e isso resultará em um tempo e despesas menores a serem despendidos com ações corretivas.

Corrêa e Gianesi (1993) elucidam que a saída do mercado de uma empresa pode ser generalizada em decorrência dos seguintes cinco pontos básicos: deficiência nas medidas de desempenho, negligência com considerações tecnológicas, especialização excessiva das funções de produção sem devida integração, perda de foco nos negócios devido a diversificação das atividades empresariais e resistência em assumir novas posturas produtivas que garantem qualidade e variedade de produtos, ou seja, a diminuição no poder de competitividade das empresas brasileiras deve-se majoritariamente à obsolescência das práticas gerenciais e tecnológicas nos sistemas produtivos. Deve-se levar em consideração a revalorização dos fatores produtivos críticos para que os objetivos estratégicos da organização sejam atingidos. A correta estruturação

do sistema de produção é fundamental para que a vantagem competitiva da empresa seja sustentada.

2.2 *Job Shop Problem* (JSP)

O agendamento da produção é uma etapa importante do PCP que tem o objetivo de organizar o sistema de produção visando a máxima eficiência operacional. Indústrias de manufatura, serviços, computação em nuvem, internet das coisas são exemplos de setores cuja esse tipo de tomada de decisão tem considerável impacto (BORDIGNON, 2022).

O JSP, formalmente, pode ser definido como a escolha da ordem de execução de um conjunto de *n jobs* em *m* máquinas visando a minimização do *makespan*, o principal indicador de produção referente ao tempo total de execução do roteiro. Esse processamento é chamado de operação e deve respeitar as seguintes delimitações: todas as máquinas estão disponíveis no instante t_0=0, cada *job* deve ser processado pelas máquinas seguindo uma ordem pré-definida, cada máquina pode processar somente um *job* por vez e assim que iniciado o mesmo não pode ser interrompido, não havendo nenhuma relação de precedência entre as operações de diferentes *jobs* (YAMADA, 2003).

De acordo com Lopes (2022), normalmente os meios de resolução de problemas de *Job Shop* são conhecidos somente por grandes empresas como IBM, HP, Microsoft e Nestlé que tem um nível de maturidade extremamente elevado com uma gestão do conhecimento que permite acesso a softwares próprios de resolução desse tipo de problema, portanto esse tópico é de total relevância tanto para empresas de grande porte quanto para empresas menores que possam empregar esse tipo de conhecimento e ainda não utilizam de nenhuma forma a PO. A diminuição dos custos da operação é capaz de propiciar uma larga vantagem competitiva para a empresa tendo em vista que esse custo que foi diminuído pode ser revertido em um aumento da margem de lucro, reinvestimento em outras áreas da empresa e também uma forma de diminuir o preço final do serviço para o consumidor, tornando o produto/serviço melhor posicionado no mercado.

2.3 Heurística

Heurísticas são algoritmos pensados para resolver problemas de modo estratégico levando em consideração a natureza do problema existente, combinando qualidade de solução com menor esforço computacional, o que reduz o tempo total de resolução (GOLDBARG *et al.*, 2016)

A heurística construtiva constrói uma solução elemento por elemento. A forma de construção varia de acordo com a função de avaliação adotada, que depende do problema a ser trabalhado. No modelo clássico, os elementos ordenados para inserção na solução primeiramente costumam ser escolhidos os que oferecem maiores benefícios, ou seja, os que contribuem mais para o objetivo de minimização ou maximização (SOUZA, 2009).

2.4 Meta-heurística

Segundo Glover e Kochenberger (2003) meta-heurísticas são métodos de resolução que harmonizam uma interação entre procedimentos de melhoria local e estratégias sapientes para escapar de ótimos locais.

Uma meta-heurística é uma ferramenta algorítmica geral, que pode ser aplicada a diferentes problemas de otimização, com modificações relativamente pequenas, para torná-la adaptável a um problema em específico. Sua utilização para resolver problemas complexos combinatorial tem sido discutida ao longo dos anos e apresentando bons resultados na resolução desse tipo de problema (SOUZA, 2008).

As meta-heurísticas utilizadas neste trabalho foram: *Iterated Local Search (ILS)* e *Variable Neighborhood Search (VNS)*. O VNS é um método de busca local que consiste em explorar o espaço de soluções através de trocas sistemáticas na estrutura de vizinhança, explorando vizinhanças cada vez mais espaçadas da solução corrente e foca em uma nova solução caso um movimento de melhoria seja realizado. O ILS é um método de busca que se utiliza de buscas locais e perturbação. Caso a busca local não seja efetiva para determinada troca de vizinhança o número de trocas aumenta.

3 METODOLOGIA

Nessa seção serão apresentados os métodos dentro da PO escolhidos para resolver o *Job Shop Problem* (JSP), a lógica utilizada para a formulação dos algoritmos e a caracterização da pesquisa.

Os métodos escolhidos foram: heurística, programação linear, meta-heurística *ILS* e *VNS*.

A metodologia utilizada é de fundamento exploratório. São formuladas hipóteses acerca do comportamento do algoritmo que influenciará no tempo de resposta e na função objetivo, utilizando o conhecimento teórico com observações prévias do comportamento do algoritmo, e em seguida essas hipóteses são testadas com o objetivo de se obter a validação da ideia. O procedimento para o desenvolvimento da pesquisa foi o de modelagem. Primeiramente, foi realizada uma pesquisa em torno 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.

3.1 Programação Linear (PL)

O modelo matemático implementado é o apresentado nas equações [3.1] à [3.7].

$$Minimizarz = \sum_{i=1}^{n} C_{im} \qquad [3.1]$$

A equação [3.1] representa a função objetivo que visa minimizar o somatório do último tempo de término do *job* i na máquina m.

$$C_{i,maq_1} \geq p_{i,maq_1} \forall i = 1, \ldots, n \qquad [3.2]$$

A equação [3.2] garante que o instante de término da tarefa i na máquina maq_1 seja maior ou igual ao tempo de processamento da tarefa i na máquina maq_1.

$$C_{i,maq_{k+1}} \geq C_{i,maq_k} + p_{i,maq_{k+1}} \qquad [3.3]$$

A equação [3.3] garante que o tempo de processamento na próxima máquina da tarefa i até dado instante vai ser maior ou igual a soma do término da tarefa i na próxima máquina com o tempo de processamento da tarefa i na máquina atual.

$$C_{jk} \geq C_{ik} + p_{jk} - M(1 - x_{ijk}) \forall i \neq j \forall i, j, k \quad [3.4]$$

$$C_{ik} \geq C_{jk} + p_{ik} - M x_{ijk} \forall i \neq j \forall i, j, k \quad [3.5]$$

O conjunto de equações [3.4] e [3.5] é disjuntivo e garante, respectivamente, se na máquina k a tarefa i precede a tarefa j ou se a tarefa j precede a tarefa i. Se $x_{ijk} = 1$ a equação [3.5] é desativada. De modo análogo, se $x_{ijk} = 0$ a equação [3.4] é desativada.

$$C_{ik} \geq 0 \forall i = 1, \ldots, n \forall k = 1, \ldots, m \quad [3.6]$$

$$x_{ijk} \in B \forall i, j, k \quad [3.7]$$

As equações [3.6] e [3.7] estabelecem os tipos de variáveis do modelo matemático. A Tabela 3.1 demonstra os resultados obtidos por Programação Linear em uma instância.

Tabela 3.1 – Resultados obtidos pelo PL na instância com 6 máquinas e 6 tarefas.

Sequência de produção	
Máquina	Tarefas
0	0 → 5 → 4 → 3 → 2 → 1
1	5 → 0 → 4 → 1 → 3 → 2
2	0 → 4 → 2 → 1 → 5 → 3
3	5 → 0 → 2 → 4 → 3 → 1
4	4 → 0 → 5 → 1 → 3 → 2
5	5 → 0 → 4 → 2 → 1 → 3

Fonte: Autoria própria

3.2 Heurística Construtiva

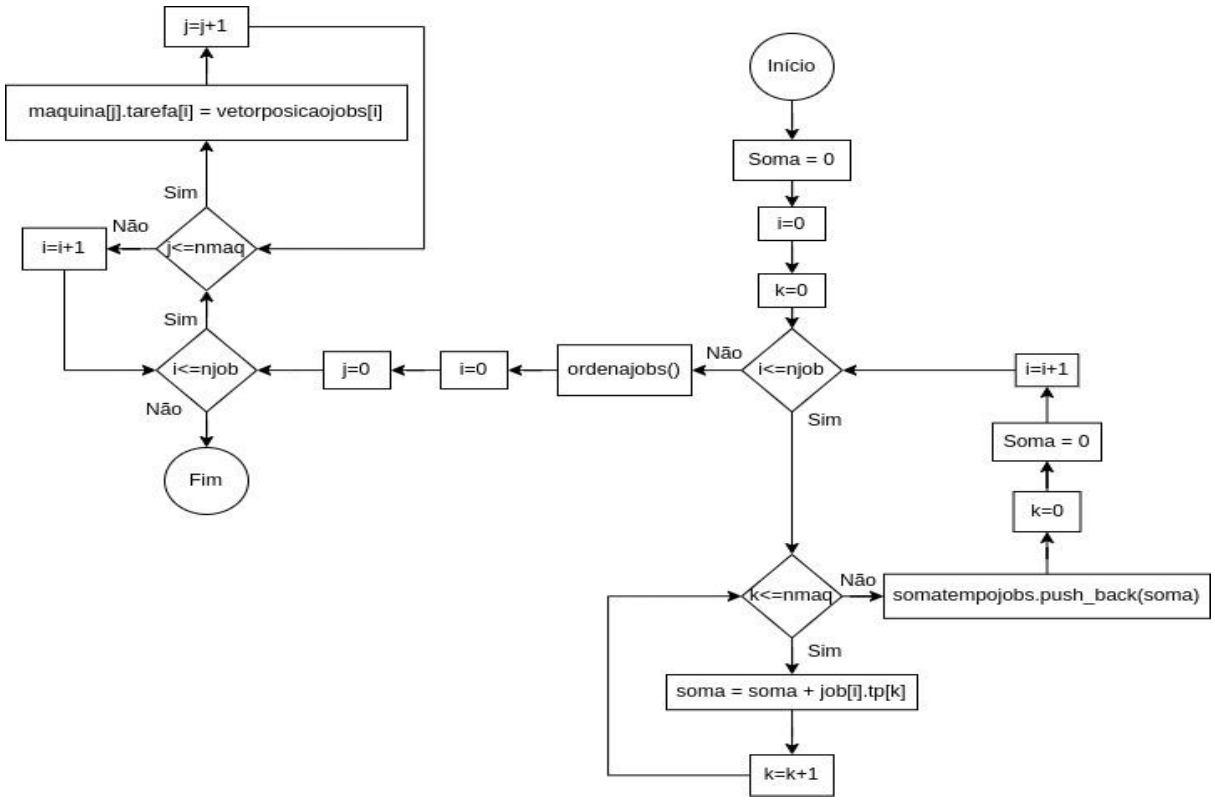
O questionamento formulado com o objetivo de melhorar as respostas encontradas pelo método aproximado foi: Dado um espaço de solução que se caracteriza por ser um conjunto das respostas viáveis para a resolução do problema, existe uma forma de montar uma solução inicial que esteja mais próxima da solução ótima nesse espaço? Caso exista, partir desse ponto resultaria em uma melhoria na resposta de todas as instâncias?

O primeiro passo para partir em busca desse novo ponto inicial foi analisar as respostas encontradas nessas instâncias utilizando a programação linear, pois ao perceber alguma lógica compartilhada entre a resposta de diferentes instâncias seria possível partir de um ponto que estivesse mais próximo da resposta ótima no espaço de solução. Vários critérios foram analisados, visando encontrar alguma forma de ordenar as tarefas se baseando no tempo de execução individual de cada *job* em sua respectiva

máquina e nos tempos totais de execução das tarefas. Na resposta final um deles demonstrou ter maior importância.

Foi constatado que os *jobs* com menor tempo total de execução tinham prioridade na fila de execução. A partir dessa constatação foi elaborado um algoritmo que monta a resposta inicial que será utilizada pela heurística que ordena as tarefas por tempo total de execução, somando o tempo de cada tarefa em todas as máquinas descritas no roteiro e os colocando em ordem crescente.

Figura 3.2 – Fluxograma da nova heurística construtiva proposta



Fonte: Autoria própria

3.2.2 Heurística de refinamento

O objetivo da heurística de refinamento é diminuir o tempo total de execução de todas as tarefas, o que implica a diminuição da função objetivo, gerando uma resposta que esteja em uma parte do espaço solução mais próxima da resposta ótima. Foi constatado que quanto maior o número de *jobs* e máquinas, maior é a taxa de melhora para algumas mudanças realizadas na resposta pela heurística de refinamento proposta

nesse trabalho. O algoritmo da heurística de refinamento funciona reformulando o roteiro de uma máquina por vez, observando quais são as tarefas que tem como primeira máquina no roteiro de execução a máquina que está sendo analisada e dando prioridade para as tarefas que na ordem de execução vem primeiro no roteiro de execução da próxima máquina a qual essa tarefa deve ser direcionada. Em caso de empate nesse critério permanece alocada primeiro a primeira tarefa analisada. Em seguida tem maior prioridade as tarefas que ficam disponíveis primeiro para a máquina analisada, sendo alocadas nessa ondem. O processo é repetido em cada máquina até que o número de tentativas de melhora seja igual ao número de máquinas, pois ao se igualarem significa que não existem mais melhorias a serem exploradas nos fatores que o algoritmo analisa. Pelo modo como as tarefas são organizadas, podendo ser feita a analogia a um guarda-roupas, essa heurística desenvolvida e proposta pelos autores nesse trabalho foi nomeada como “Heurística do Guarda-roupa”.

Figura 3.4 – Algoritmo Heurística do Guarda Roupa

Heurística do Guarda-Roupa
1. t←0;
2. Ordena os trabalhos com base no tempo total de execução;
3. enquanto (os critérios de parada não estiverem satisfeitos) faça
4. Separa os trabalhos em duas categorias na máquina t: Disponíveis e não disponíveis
5. Cria um vetor de trabalhos não disponíveis ordenados com base no tempo de finalização na máquina anterior à máquina t;
6. Cria um vetor de trabalhos disponíveis ordenados com base na posição de execução na próxima máquina do roteiro;
7. Junta os dois vetores, sendo o primeiro o vetor dos trabalhos disponíveis e esse resultado se torna o novo roteiro;
8. t = t + 1;
9. **fim-enquanto;**
Fim Heurística do Guarda Roupa

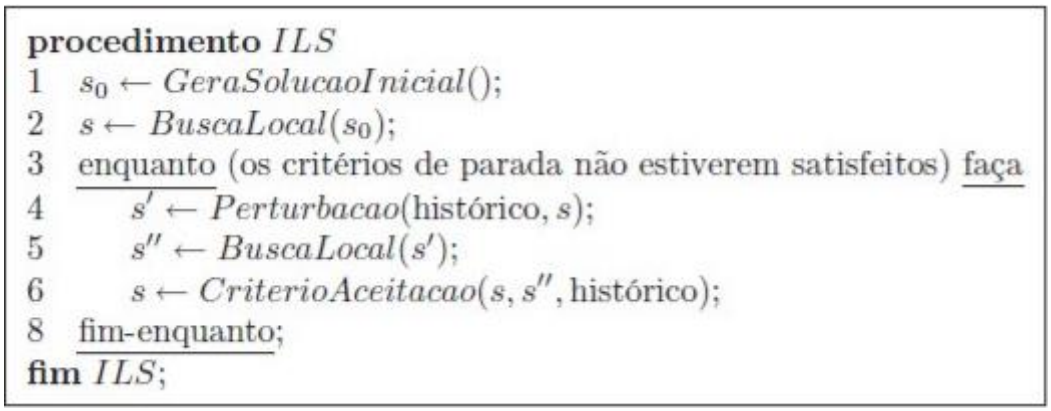
Fonte: Autoria própria

3.3 *Iterated Local Search* (ILS)

Meta-heurísticas são métodos subsequentes com o objetivo de encontrar uma solução que se aproxime o máximo possível da resposta da programação linear em um tempo inferior, eventualmente é possível chegar à solução ótima. 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).

Figura 3.5 - Algoritmo Iterated Local Search



Fonte: Souza (2009)

A meta-heurística *Iterated Local Search* (ILS) se caracteriza por construir uma sequência de soluções geradas por uma heurística incorporada. A lógica por trás da ILS é que os ótimos locais de um problema de otimização podem ser encontrados a partir de perturbações na solução ótima local corrente (COELHO *et al.*, 2018).

O algoritmo ILS realiza um procedimento de busca local que pode ser otimizado a partir da geração de novas soluções de partida obtidas por meio de perturbações na solução ótima local. Um procedimento de busca local é realizado para todas as máquinas e todas as tarefas, utilizando um mecanismo de troca que consiste em selecionar uma máquina e após essa seleção escolhe duas tarefas e as troca entre si no roteiro de execução dessa determinada máquina. Caso ocorra a diminuição da função objetivo após a realização desse processo a menor função objetivo encontrada é atualizada. Em seguida ocorre o processo de perturbação, que possui três níveis de intensidade e realiza sucessivas alterações aleatórias na resposta sorteando uma máquina e duas tarefas do roteiro de sequência da máquina, que serão trocadas entre si na sequência de realização, gerando uma nova resposta para que uma nova Função Objetivo (FO) seja calculada. Caso nenhuma melhora seja encontrada em determinado nível de perturbação o próximo nível existente é acionado. Caso nenhuma melhora ocorra no último nível de perturbação esse processo é encerrado. O processo de busca local também ocorre dentro do processo de perturbação após o sorteio aleatório. Na perturbação nível um, duas

máquinas são sorteadas, enquanto na perturbação nível dois, três máquinas são sorteadas e por fim, na perturbação nível três, quatro máquinas são sorteadas. Após a finalização do processo de perturbação ocorre novamente uma busca local no fim da iteração.

3.4 Variable Neighborhood Search (VNS)

A busca em vizinhança variável foi proposta no trabalho de Mladenović e Hansen (1997), que consiste em explorar o espaço solução trocando sistematicamente as estruturas de vizinhança. O método busca vizinhanças cada vez mais distantes, focalizando a busca em torno de uma nova solução, apenas se ocorre a melhora da solução. Dada uma solução inicial, que pode ser construída de qualquer forma, é buscado um vizinho aleatório s' dentro da vizinhança N_K da solução corrente. O vizinho é submetido a um processo de busca local. Se a solução s'' for melhor que a solução corrente, a busca continua recomeçando da primeira vizinhança. Se a solução s'' não for melhor que a solução corrente a busca continua a partir da estrutura de vizinhança N_{K+1}. A solução s' é gerada porque nesse ponto do algoritmo uma regra determinística não seria capaz de varrer o espaço solução de uma maneira eficiente. O algoritmo pode ter diferentes critérios de parada, tais como tempo de execução, número máximo de iterações consecutivas entre dois melhoramentos ou número máximo de iterações.

Figura 3.6 - Algoritmo *Variable Neighborhood Search*

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

Fonte: Souza (2009)

O método VNS consiste em buscar a melhora da solução atual utilizando uma estrutura de vizinhança. A vizinhança corrente é trocada pela próxima quando a atual não possibilita melhora na função objetivo. O término do algoritmo é encontrado quando não há mais a possibilidade de melhorar a solução atual após o uso de todas as estruturas de vizinhança (CONSERVA,2022)

O algoritmo VNS realiza sucessivas alterações aleatórias na melhor resposta encontrada sorteando uma máquina e duas tarefas do roteiro de sequência da máquina, que serão trocadas entre si na sequência de realização, gerando uma nova resposta e a partir disso a nova Função Objetivo (FO) é calculada. Caso ocorra a diminuição da função objetivo após essa alteração a melhor resposta encontrada é atualizada. Independe da melhoria da resposta também ocorre uma busca local para todas as máquinas e tarefas que utiliza o mesmo mecanismo de troca das estruturas de vizinhança. Foram utilizadas 6 estruturas de vizinhança, sendo a diferença entre elas o número de máquinas sorteadas aleatoriamente para que duas tarefas do roteiro fossem trocadas entre si em cada máquina sorteada. O critério de parada utilizado foi o número máximo de 5.000 iterações.

4 RESULTADOS

Conforme citado na metodologia do trabalho, o primeiro passo foi o estudo sobre o tema tratado e a escolha das instâncias iniciais do teste no banco de dados online OR - Library. As instâncias foram selecionadas de acordo com o problema, visando englobar desde problemas pequenos até problemas grandes. Foram selecionadas sete instâncias do OR – Library e foi utilizado um notebook Intel core i7 7ª geração com memória RAM de 8GB.

4.1 OR-Library

As instâncias foram selecionadas de acordo com o problema, visando englobar desde problemas pequenos até problemas grandes. Foram selecionadas 7 instâncias de teste da OR-Library com os tempos de execução já definidos, as quais são descritas na Tabela 4.1.

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

Instância de teste	Máquinas	Tarefas
1	6	6
2	10	5
3	10	10
4	20	10
5	11	5
6	30	10
7	50	10

Fonte: Autoria própria

4.2 Comparação dos resultados com VNS

Os resultados da Função Objetivo (FO) e os tempos de execução do algoritmo correspondente às diferentes instâncias utilizando a programação linear, a heurística construtiva de Baker e Trietsch a nova heurística construtiva proposta (heurística do guarda-roupa) são apresentados na Tabela 4.2.

Tabela 4.2 – Resultados obtidos com limite de 5.000 iterações no VNS

Instância de teste	Programação Linear		Heurística de Baker e Trietsch		Heurística de ordenação		Heurística do Guarda-roupa	
	Tempo (s)	FO	Tempo (s)	FO	Tempo (s)	FO	Tempo (s)	FO
6x6	1	265	10	296	8	265	8	265
10x5	1.508	4.153	32	4.588	26	4289	29	4.354
10x10	86.400	7.505	115	9.792	97	7.713	99	7.844
20x10	86.400	1.774	1.546	2.697	1.506	1.967	1.444	1.909
11x5	14.544	47.400	56	62.863	57	51.558	59	51.558
30x10	86.400	37.510	6.020	47.038	3.899	56.621	4.648	44.822
50x10	86.400	97.762	32.231	115.064	29.008	161.036	30.012	112.693

Fonte: Autoria própria

Após implementar a metodologia proposta e comparar os resultados da heurística de Baker e Trietsch com a heurística de ordenação foi possível concluir que a instância 6x6, a menor de todas, conseguiu reduzir a função objetivo de 296 para 265 (redução de 10,47%), sendo a nova resposta a mesma resposta encontrada utilizando a programação linear porém em um tempo computacional um pouco maior. A função objetivo da instância 10x5 foi reduzida de 4.588 para 4.289 (redução de 6,5%) enquanto o tempo diminuiu de 32 para 26 segundos (redução de 18,75%).

A função objetivo da instância 10x10 foi minimizada de 9.792 para 7.713 (redução de 21,23%) enquanto o tempo diminuiu de 115 para 97 segundos (redução de 15,65%). A função objetivo da instância 20x10 foi reduzida de 2.697 para 1.967 (redução de 27%) enquanto o tempo diminuiu de 1.546 para 1.506 segundos (redução de 2,58%). A função objetivo da instância 11x5 foi minimizada de 62.863 para 51.558 (redução de 17,98%) enquanto o tempo aumentou em um segundo.

As instâncias 30x10 e 50x10 não tiveram os seus valores minimizados na função objetivo, porém é possível observar uma redução de 35% e 10% no tempo de execução, respectivamente, devido ao fato das primeiras estruturas de vizinhança serem de mais

rápido processamento, ou seja, quanto mais melhorias ocorrerem durante a execução do algoritmo mais essas primeiras estruturas são utilizadas, acarretando a diminuição do tempo de processamento. Todas as outras instâncias alcançaram reduções tanto na função objetivo quanto no tempo de execução do algoritmo.

Utilizar somente a heurística construtiva proposta não se mostrou completamente eficaz na diminuição da função objetivo para as últimas duas instâncias pois dado seu tamanho utilizar essa lógica de montagem afasta muito o ponto inicial do ponto ótimo pretendido no espaço solução, o que faz com que a aproximação não seja eficiente. Utilizar a heurística de refinamento, que prioriza as tarefas que iniciam em determinada máquina e em seguida as próximas tarefas a medida em que vão ficando disponíveis, permite partir de um ponto no espaço solução o qual é possível obter uma aproximação satisfatória para essas instâncias.

Comparando os resultados da resposta inicial montada pela heurística construtiva de Baker e Trietsch com os resultados produzidos pela heurística de ordenação em conjunto com a heurística do guarda-roupa a função objetivo da instância 6x6 reduziu de 296 para 265 (redução de 10,47%) enquanto o tempo diminuiu de 10 para 8 segundos (redução de 20%).A função objetivo da instância 10x5 foi minimizada de 4.588 para 4.354 (redução de 5%) enquanto o tempo reduziu de 32 para 29 segundos (redução de 9,4%).

A função objetivo da instância 10x10 reduziu de 9.792 para 7.844 (redução de 19,9%) enquanto o tempo diminuiu de 115 para 99 segundos (redução de 13,9%). A função objetivo da instância 20x10 foi minimizada de 2.697 para 1.909 (redução de 29,2%) enquanto o tempo reduziu de 1.546 para 1.444 segundos (redução de 6,6%). A função objetivo da instância 11x5 reduziu de 62.865 para 51.558 (redução de 18%) enquanto o tempo aumentou de 56 para 59 segundos (acréscimo de 5,4%).

A função objetivo da instância 30x10 reduziu de 47.038 para 44.812 (redução de 4,7%) enquanto o tempo diminuiu de 6.020 para 4.648 segundos (redução de 22,7%). A função objetivo da instância 50x10 foi minimizada de 115.064 para 112.693 (redução de 2%) enquanto o tempo diminuiu de 32.231 para 30.012 segundos (redução de 6,9%).

Comparando os resultados produzidos pela heurística de ordenação com a resposta produzida pela heurística de ordenação em conjunto com a heurística do guarda-roupa a função objetivo da instância 6x6 manteve-se em 265, mesma resposta encontrada pela programação linear, enquanto o tempo de execução também se manteve

o mesmo. A função objetivo da instância 10x5 aumentou de 4.289 para 4.354 (acréscimo de 1,5%) enquanto o tempo de execução aumentou de 26 para 29 segundos (acréscimo de 11,53%). A função objetivo da instância 10x10 aumentou de 7.713 para 7.844 (acréscimo 1,7%) enquanto o tempo aumentou de 97 para 99 segundos (acréscimo de 2%).

A função objetivo da instância 20x10 obteve uma redução de 1.967 para 1.909 (redução de 3%) enquanto o tempo de execução diminuiu de 1.506 para 1.444 (redução de 4%). A função objetivo da instância 11x5 manteve-se inalterada enquanto o tempo de execução aumentou de 57 para 59 segundos (acréscimo de 3%). A função objetivo da instância 30x10 reduziu de 56.621 para 44.822 (redução de 20,84%) enquanto o tempo aumentou de 3.899 para 4.648 segundos (acréscimo de 19,21%). Nesse caso a função objetivo da instância 50x10 foi minimizada de 161.036 para 112.693 (redução de 30%) enquanto o tempo aumentou de 29.008 para 30.012 segundos (aumento de 3,5%).

Ao comparar os resultados obtidos pela programação linear, que é o método que produz os menores valores da função objetivo por ser um método exato capaz de encontrar a solução ótima, com os resultados obtidos pela heurística de ordenação é possível concluir que com exceção das instâncias 30x10 e 50x10 foi a heurística apresentada nesse trabalho que mais se aproximou desses resultados.

Para a instância 6x6 não houve diferença percentual pois o mesmo resultado foi encontrado em um tempo 7 segundos maior. Para a instância 10x5 a diferença percentual foi de 3,17% em um tempo 98% menor. Para a instância 10x10 a diferença percentual foi de 2,7% em um tempo 99,9% menor. Para a instância 20x10 a diferença percentual foi de 9,8% em um tempo 98,2% menor. Para a instância 11x5 a diferença percentual foi de 8% em um tempo 99,6% menor.

Ao comparar os resultados obtidos pela programação linear com os resultados obtidos pela heurística de ordenação em conjunto com a heurística do guarda-roupa é possível concluir que esse foi o melhor método apresentado nesse trabalho para resolver instâncias dessa magnitude. Para a instância 30x10 a diferença percentual foi de 18% para um tempo 96,9% menor. Para a instância 50x10 a diferença percentual foi de 13,2% para um tempo 65,2% menor.

4.3 Comparação dos resultados com ILS

Os resultados da Função Objetivo (FO) e os tempos de execução do algoritmo correspondente às diferentes instâncias utilizando a programação linear, a heurística construtiva de Baker e Trietsch a nova heurística construtiva proposta (heurística do guarda-roupa) são apresentados na Tabela 4.3.

Tabela 4.3 – Resultados obtidos com limite de 5.000 iterações no ILS

Instância de teste	Programação Linear		Heurística de Baker e Trietsch		Heurística de ordenação		Heurística do Guarda-roupa	
	Tempo (s)	FO	Tempo (s)	FO	Tempo (s)	FO	Tempo (s)	FO
6x6	1	265	5	280	3	280	4	283
10x5	1.508	4.153	28	4363	12	4272	12	4272
10x10	86.400	7.505	72	9673	46	8138	45	8159
20x10	86.400	1.774	2186	2737	477	1981	479	1939
11x5	14.544	47.400	30	56702	19	51558	20	51558
30x10	86.400	37.510	10547	45204	1897	48301	2071	48518
50x10	86.400	97.762	96679	107522	14531	195846	15355	113936

Fonte: Autoria própria

Após implementar a metodologia proposta e comparar os resultados da heurística de Baker e Trietsch com a heurística de ordenação foi possível concluir que a instância 6x6, a menor de todas, chegaram a uma função objetivo de 280, enquanto o tempo caiu de 5 para 3 segundos (redução de 40%) A função objetivo da instância 10x5 foi reduzida de 4.363 para 4.272 (redução de 2%) enquanto o tempo diminuiu de 28 para 12 segundos (redução de 57%). A função objetivo da instância 10x10 foi minimizada de 9.673 para 8.138 (redução de 15,9%) enquanto o tempo diminuiu de 72 para 46 segundos (redução de 36%). A função objetivo da instância 20x10 foi reduzida de 2.737 para 1.981 (redução de 27,6%) enquanto o tempo diminuiu de 8.186 para 477 segundos (redução de 78%). A função objetivo da instância 11x5 foi minimizada de 56.702 para 51.558 (redução de 9%) enquanto o tempo diminuiu de 30 para 19 segundos (redução de 36%).

As instâncias 30x10 e 50x10 não tiveram os seus valores minimizados na função objetivo, porém é possível observar uma redução de 82% e 84% no tempo de execução, respectivamente, devido ao fato da perturbação de nível um ser de mais rápido processamento, ou seja, quanto mais melhorias ocorrerem durante a execução do algoritmo mais esse nível de perturbação é utilizado, acarretando a diminuição do tempo de processamento. Todas as outras instâncias alcançaram reduções tanto na função objetivo quanto no tempo de execução do algoritmo.

Utilizar somente a heurística construtiva proposta não se mostrou completamente eficaz na diminuição da função objetivo para as últimas duas instâncias pois dado seu tamanho utilizar essa lógica de montagem afasta muito o ponto inicial do ponto ótimo pretendido no espaço solução, o que faz com que a aproximação não seja eficiente. Utilizar a heurística de refinamento, que prioriza as tarefas que iniciam em determinada máquina e em seguida as próximas tarefas a medida em que vão ficando disponíveis, permite partir de um ponto no espaço solução o qual é possível obter uma aproximação satisfatória para essas instâncias.

Comparando os resultados da resposta inicial montada pela heurística construtiva de Baker e Trietsch com os resultados produzidos pela heurística de ordenação em conjunto com a heurística do guarda-roupa a função objetivo da instância 6x6 aumentou de 280 para 283 (acréscimo de 1%) enquanto o tempo diminuiu de 5 para 4 segundos (redução de 20%). A função objetivo da instância 10x5 foi minimizada de 4.363 para 4.272 (redução de 2%) enquanto o tempo reduziu de 28 para 12 segundos (redução de 57%).

A função objetivo da instância 10x10 reduziu de 9.673 para 8159 (redução de 15,7%) enquanto o tempo diminuiu de 72 para 45 segundos (redução de 37,5%). A função objetivo da instância 20x10 foi minimizada de 2.737 para 1.939 (redução de 29%) enquanto o tempo reduziu de 2.186 para 479 segundos (redução de 78%). A função objetivo da instância 11x5 reduziu de 56.702 para 51.558 (redução de 9%) enquanto o tempo diminuiu de 30 para 20 segundos (redução de 33%).

A função objetivo da instância 30x10 aumentou de 45.204 para 48.518 (acréscimo de 7,3%) enquanto o tempo diminuiu de 10.547 para 2.071 segundos (redução de 80,4%). A função objetivo da instância 50x10 aumentou de 107.552 para 113.936 (acréscimo de 5,9%) enquanto o tempo diminuiu de 96.679 para 15.355 segundos (redução de 84%).

Comparando os resultados produzidos pela heurística de ordenação com a resposta produzida pela heurística de ordenação em conjunto com a heurística do guarda-roupa a função objetivo da instância 6x6 subiu 280 para 283 (acréscimo de 1%) enquanto o tempo de execução aumentou em um segundo. A função objetivo da instância 10x5 manteve-se em 4.272 enquanto o tempo de execução também se manteve em 12 segundos. A função objetivo da instância 10x10 aumentou de 8.138 para 8.159 (acréscimo 0,25%) enquanto o tempo de execução caiu em um segundo.

A função objetivo da instância 20x10 obteve uma redução de 1.981 para 1.939 (redução de 2,1%) enquanto o tempo de execução aumentou de 477 para 479 (acréscimo de 0,4%). A função objetivo da instância 11x5 manteve-se inalterada enquanto o tempo de execução aumentou em um segundo. A função objetivo da instância 30x10 aumentou de 48.301 para 48.518 (acréscimo de 0,4%) enquanto o tempo aumentou de 1.897 para 2.071 segundos (acréscimo de 9,2%). Nesse caso a função objetivo da instância 50x10 foi minimizada de 195.846 para 113.936 (redução de 41,8%) enquanto o tempo aumentou de 14.531 para 15.355 segundos (acréscimo de 5,7%).

Ao comparar os resultados obtidos pela programação linear, que é o método que produz os menores valores da função objetivo por ser um método exato capaz de encontrar a solução ótima, com os resultados obtidos pela heurística de ordenação é possível concluir que com exceção das instâncias 30x10 e 50x10 foi a heurística apresentada nesse trabalho que mais se aproximou desses resultados.

Para a instância 6x6 a diferença percentual foi de 5,4% em um tempo dois segundos maior. Para a instância 10x5 a diferença percentual foi de 2,8% em um tempo 99,2% menor. Para a instância 10x10 a diferença percentual foi de 7,8 em um tempo 99,9% menor. Para a instância 20x10 a diferença percentual foi de 10,4% em um tempo 99,4% menor. Para a instância 11x5 a diferença percentual foi de 8% em um tempo 99,9% menor.

Ao comparar os resultados obtidos pela programação linear com os resultados obtidos pela heurística de ordenação em conjunto com a heurística do guarda-roupa é possível concluir que esse foi o melhor método apresentado nesse trabalho para resolver instâncias dessa magnitude pois apesar da função objetivo ter sido um pouco prejudicada, em relação a heurística de Baker e Trietsch, a melhoria no tempo de execução do algoritmo foi significativa. Para a instância 30x10 a diferença percentual foi de 22,7% para

um tempo 97,6% menor. Para a instância 50x10 a diferença percentual foi de 14,2% para um tempo 82,2% menor.

5. Conclusão

Durante o desenvolvimento da pesquisa, foram exploradas diversas heurísticas e técnicas para a resolução do Job Shop Problem, com o objetivo de diminuir o tempo de solução e obter resultados que se aproximassem ao máximo das soluções obtidas utilizando a programação linear. Os objetivos específicos foram abordados de maneira metódica e sistemática, resultando em avanços significativos na resolução do problema. O modelo matemático do Job Shop Problem foi implementado com sucesso, e várias técnicas foram aplicadas, incluindo a meta-heurística Variable Neighborhood Search, heurísticas construtivas, heurísticas de refinamento e a meta-heurística Iterated Local Search.

A comparação dos resultados obtidos com cada uma dessas abordagens permitiu uma análise criteriosa e a identificação da melhor forma de resolver o problema. Os dados coletados e as análises realizadas forneceram *insights* valiosos sobre as vantagens e desvantagens de cada técnica, bem como suas aplicações práticas. Portanto, com base nos resultados alcançados e nas conclusões derivadas deste estudo, é seguro afirmar que os objetivos propostos foram cumpridos com êxito. Este trabalho contribuiu para o avanço no campo da otimização de problemas de programação da oficina e oferece uma base sólida para futuras pesquisas nessa área.

REFERÊNCIAS

VEGGIAN, Viviane Amaro; SILVA, TF da. Planejamento e controle da produção. FAEF–Revistas Eletrônicas, 2011.

TUBINO, Dalvio Ferrari. Planejamento e controle da produção: teoria e prática. São Paulo: Atlas, 2007. 190 p.

CORRÊA, H. L.; GIANESI, I. G. N. Just in Time, MRPII e OPTC: Um enfoque estratégico. 2. ed. São Paulo: Atlas, 1993.

GUERRINI, Fabio Muller; BELHOT, Renato Vairo; JUNIOR, Walter Azzolini. Planejamento e controle da produção. Projeto e operação de sistemas. 1ª Ed. Editora Elsevier. Rio de Janeiro, 2014.

BORDIGNON, Jonathan Farias et al. Estudo sobre as operações críticas no agendamento de tarefas em sistemas produtivos do tipo job shop. Research, Society and Development, v. 11, n. 5, p. e17111528035-e17111528035, 2022.

Yamada, T. (2003). Studies on metaheuristics for jobshop and flowshop scheduling problems.
PhD dissertação Kyoto University. http://www.kecl.ntt.co.jp/as/members/yamada/YamadaThesis.pdf

GOLDBARG, E.; GOLDBARG, M.; LUNA, H. Otimização combinatória e metaheurísticas: algoritmos e apliacações. [S. l.]: Elsevier Brasil, 2016.

CONSERVA, Júlio Cesar Vasconcelos et al. Abordagem metaheurística híbrida para minimização de custos por antecipação e atrasos na produção. 2022.

SANTOS, Marcos. Utilização do algoritmo branch and bound na otimização da produção de uma indústria de produtos plásticos. Pesquisa Operacional, Rio de Janeiro, Revista de Trabalhos Acadêmicos Lusófona, v.2. n.2. Abr./Jun.2019.

REIS, Liliane. Mix ótimo de aquisições de ingredientes para uma microempresa do setor de food service. Pesquisa Operacional, Paraná, Congresso Brasileiro de Engenharia de Produção, p.35 42. Disponível em: https://aprepro.org.br/conbrepro/2020/anais/arquivos/09252020_080943_5f6ddb37c052e.pdf. Acesso em: 25 mar. 2021.

MENEZES, Ivana. Determinação do mix de produto via programação linear: Estudo de caso de um laticínio na cidade de Bambuí - MG. Pesquisa Operacional, Gestão da Produção em Foco– Volume 18/ Organização Editora Poisson – Belo Horizonte - MG: Poisson, 2018 258p. 2018.

ARENALES, M., ARMENTANDO, V., MORABITO, R., YANASSE, H. (2007). Pesquisa Operacional. Rio de Janeiro: Elsevier. CAETANO, D. Pesquisa operacional: introdução a modelagem matemática. 2008, p. 1-203.

SCOFIELD, Wendrer Carlos. Aplicação de Algoritmos Genéticos ao Problema Job- Shop. Orientador: Marcone Jamilson. 2002. 131 f. TCC (Graduação) – Bacharelado em Ciência da Computação, Departamento de Computação do Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto.

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

SILVA, Janaina. Uma meta-heurística baseada em busca em vizinhança variável para o problema Job Shop. Tese (Bacharelado em Engenharia de Produção) – Faculdade de Ciências Integradas do Pontal - FACIP, Universidade Federal de Uberlândia. Ituiutaba, jun. 2016. Disponível em: https://repositorio.ufu.br/handle/123456789/20367. Acesso em: 25 mar. 2021.

Marcone Jamilson Freitas Souza. "Inteligência Computacional para Otimização". Apostila de sala de aula. Inteligência Computacional para Otimização. (Professor Marcone Jamilson Freitas Souza.) Departamento de Computação, Instituto de Ciências Exatas e Biológicas, Universidade Federal de Ouro Preto. Jan. de 2009. Digital.

MLADENOVIC, N.; HANSEN, P. Variable neighborhood Search. Computers and Operations Research, v. 24, n. 11, p. 1097–1100. 1997.

GLOVER, F.; KOCHENBERGER, G. A. Handbook of metaheuristics. Nova Iorque: Kluwer, 2003. 557 p.

SILVA, Carlos Alexandre; DE SOUZA, Sérgio Ricardo. Uma Aplicação da Meta-heurística Híbrida Simulated Annealing-Iterated Local Search ao Problema de Fluxo Multiproduto sob o Espaço Capacitado. Trends in Computational and Applied Mathematics, v. 9, n. 1, p. 165-174, 2008.

LOPES, Ana Lúcia Miranda. Pesquisa operacional. 2022.

HENRIQUES, BÁRBARA BUENO. Pesquisa operacional voltada para o empreendedorismo e inovação em micro empresa. Monografia, Universidade Estadual de Campinas, Campinas, 2022.

GLOSSÁRIO

Heurística: É uma abordagem prática que utiliza regras simplificadas lógicas para solucionar problemas complexos, equilibrando eficiência e qualidade dos resultados em cenários onde a solução ótima não é alcançável ou viável em tempo hábil.

ILS: Meta-heurística que combina busca local intensiva com perturbação e diversificação em múltiplas iterações para resolver problemas de otimização. Ela permite a refinamento gradual da solução por meio de busca local e introduz diversificação por meio de perturbações controladas.

Job Shop: É um arranjo produtivo caracterizado pela execução sequencial de diferentes tarefas em estações de trabalho específicas.

Meta-heurística: Algoritmos de utilizam estratégias não específicas de explorar de forma mais ampla o espaço das soluções.

Programação Linear: Técnica matemática que visa encontrar a solução ótima para um problema de otimização linear, no qual se busca maximizar ou minimizar uma função objetivo sujeita a um conjunto de restrições lineares.

VNS: Meta-heurística que explora diferentes vizinhanças de soluções em busca de melhores resultados em problemas de otimização combinatória. Ela combina intensificação e diversificação da busca, permitindo escapar de mínimos locais e explorar diferentes regiões do espaço de busca.

APÊNDICE A - <CÓDIGO FONTE>

/*Copyright 2017 Jorge von Atzingen dos Reis
Copyright 2018 Bruna Bernardes de Aguiar e Jorge von Atzingen dos Reis
Copyright 2020 Laura Caligaris Perim Morelli e Jorge von Atzingen dos Reis
Copyright 2021 Adriel Henrique Beato de França e Jorge von Atzingen dos Reis

Este programa é um software livre; você pode redistribuí-lo e/ou modificá-lo sob os termos da Licença Pública Geral GNU como publicada pela Fundação do Software Livre (FSF); na versão 3 da Licença, ou (a seu critério) qualquer versão posterior.

Este programa é distribuído na esperança de que possa ser útil, mas SEM NENHUMA GARANTIA; sem uma garantia implícita de ADEQUAÇÃO a qualquer MERCADO ou APLICAÇÃO EM PARTICULAR. Veja a Licença Pública Geral GNU para mais detalhes.

Você deve ter recebido uma cópia da Licença Pública Geral GNU junto com este programa. Se não, veja <http://www.gnu.org/licenses/>.*/

```
//-------------------------------------------------------------------------
//        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
//-------------------------------------------------------------------------
/*Ci5-8250U, 8GB
        Dimensão -->   Tempo  --> FO
1-PL            06x06 -->   0.35s    -->   265 ut
2-HC            06x06 -->   0.01s    -->   316 ut
3-VNS           06x06 -->   8s              -->   296 ut
4-PL/VNS        06x06 -->   68s       -->   265 ut
5-ILS           06x06 -->   3s              -->   283 ut
```

```
6-PL/ILS       06x06 -->   64s      -->   265 ut
7-BT           06x06 -->   3s              -->   307 ut
8-PL/ILS       06x06 -->   62s      -->   265 ut
9-VNS/BT       06x06 -->   11s      -->   295 ut*/
//--------------------------------------------------------------------------
//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
#include <random>
using namespace std;

//--------------------------------------------------------------------------
//Declaração das Constantes
//--------------------------------------------------------------------------

//Constantes gerais do programa
#define M 99999          //número suficientemente grande
#define MAX_JOB 50  //número máximo de tarefas
#define MAX_MAQ 10 //número máximo de máquinas

//Constantes do BT
#define ITERMAXBT 5000  //número máximo de iterações do método Busca Tabu
#define TEMPOMAX 120    //tempo máximo de execução do método Busca Tabu
#define TAMMAX_LISTA 20        //tamanho máximo da lista TABU //TROCA A LISTA
AQUI

//Constantes do ILS
#define ITERMAXILS 5000   //número máximo de iterações do método ILS
```

```
#define TEMPOMAXILS 120  //tempo máximo de execução do método ILS

//Constantes do SA
#define temperatura 100000; //Temperatura inicial do método Simulated Annealing
#define SAMax 9000 //Número máximo de iterações antes da temperatura diminuir no método Simulated Annealing
#define alfa 0.99 //Taxa de decaimento da temperatura do método Simulated Annealing

//Constantes do VNS
#define ITERMAXVNS 5000 //número máximo de iterações do método VNS
#define TEMPOMAXVNS 120 //tempo máximo de execução do método VNS
#define KMAX 6  //número máximo de estruturas de vizinhança
#define KMAXMODIFICADO 7  //número máximo de estruturas de vizinhança


//-------------------------------------------------------------------------
//Declaração 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
float x; // Variável aletória utilizada na função simannealing
int contador=0;

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

//Busca TABU
struct lista
{
        int i, j, maq;
        struct lista *proximo;  /* ponteiro para a próxima entrada */
        struct lista *anterior; /* ponteiro para o registro anterior */
};
struct lista *iniciotabu, *finaltabu, *tabu; //lista tabu
```

```
int melhor_i, melhor_j, melhor_maq;

//-------------------------------------------------------------------------
//Declaração das funções
//-------------------------------------------------------------------------
void buscalocal(int s);
void calculafo(int s);
void calculatempo(int s);
void formatacao();
void heuristicaconstrutiva();
void imprime(int s);
void jobshopgurobi(int s);
void leia();
int menu();

//VNS e ILS
void ils();
void perturbacao();
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();
void vnsmodificado();

//BT
void apaga_registro_tabu(struct lista *registro, struct lista **iniciotabu, struct lista
**finaltabu);
void buscatabu();
void display_lista_tabu(struct lista **iniciotabu, struct lista **finaltabu);
bool esta_lista_tabu(int i, int j, int maq, struct lista **iniciotabu);
void insere_lista_tabu(struct lista *i, struct lista **iniciotabu, struct lista **finaltabu);
```

```
int lenght_lista_tabu(struct lista **iniciotabu, struct lista **finaltabu);

//VNS e BT
void vnsbuscatabu();

//Simulated Annealing
void simannealing();
void respostapseudoaleatoria(int x, int s);
void respostaordenatempo();
void menorrespostaaleatoria(int x, int s);
void heuristicaderefinamento();
//---------------------------------------------------------------------------
//Programa principal
//---------------------------------------------------------------------------
int main()
{
	//srand(1000);			//utiliza o número 1000 como semente de números
aleatórios
	srand(time(NULL)); //utiliza a hora do relógio como semente

	int escolha;
	do{
		escolha=menu();
		switch (escolha)
		{
			case 1:		//Programação Linear
				leia();
				jobshopgurobi(0);
				imprime(0);
			break;

			case 2:		//Heurística Construtiva
				leia();
				heuristicaconstrutiva();
```

```
                imprime(1);
            break;

            case 3:     //Busca em Vizinhança Variável
                leia();
                respostaordenatempo();
                //menorrespostaaleatoria(MAX_MAQ,MAX_JOB);
                //heuristicaconstrutiva();
                //respostapseudoaleatoria(MAX_MAQ,MAX_JOB);
                vns();
                imprime(1);
            break;

case 4: //PL com VNS
                leia();
                jobshopgurobi(1);
   formatacao();
                vns();
                imprime(1);
            break;

            case 5:     //Iterated Local Search
                leia();
                //heuristicaconstrutiva();
                respostaordenatempo();
                heuristicaderefinamento();
                ils();
                imprime(1);
            break;

            case 6: //PL com ILS
                leia();
                jobshopgurobi(1);
   formatacao();
```

```
                ils();
                imprime(1);
            break;

            case 7:     //Busca Tabu
                leia();
                heuristicaconstrutiva();
                buscatabu();
                imprime(1);
            break;

            case 8: //PL com BT
                leia();
                jobshopgurobi(1);
formatacao();
                buscatabu();
                imprime(1);
            break;

            case 9:     //VNS com Busca Tabu
                leia();
                heuristicaconstrutiva();
                vnsbuscatabu();
                imprime(1);
            break;
            case 10:    //Simulated Annealing
                leia();
                //heuristicaconstrutiva();
                //respostapseudoaleatoria(MAX_MAQ,MAX_JOB);
                respostaordenatempo();
                //respostacerteira();
                simannealing();
                imprime(1);
            case 11:    //VNS Melhorado
```

```
                                leia();
                                respostaordenatempo();
                                vnsmodificado();
                                imprime(1);

                        break;

                }
        }while(escolha);
        return 0;
}

//------------------------------------------------------------------------
//menu: menu de escolha
//------------------------------------------------------------------------
int menu()
{
        int choice;
        do{
                cout<<"\n------------------\n";
                cout<<"1- Programação Linear\n";
                cout<<"2- Heurística Construtiva\n";
                cout<<"3- Busca em Vizinhança Variável\n";
      cout<<"4- PL com VNS\n";
      cout<<"5- Iterated Local Search\n";
      cout<<"6- PL com ILS\n";
      cout<<"7- Busca Tabu\n";
      cout<<"8- PL com BT\n";
      cout<<"9- VNS com Busca Tabu\n";
      cout<<"10- Simulated Annealing\n";
      cout<<"11- VNS modificado\n";
                cout<<"0- Sair.\n";
                cout<<"------------------\n";
                cin>>choice;
```

```
        }while(choice<0||choice>11);
        return choice;
}

//------------------------------------------------------------------------
//Função leia: Lê 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.tempofim[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 resolução 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, 3600);//Tempo limitado
a 3.600s
         break;

         case 1:
            model.getEnv().set(GRB_DoubleParam_TimeLimit, 4898.87);//Tempo
limitado a 60s
            model.getEnv().set(GRB_DoubleParam_MIPGap, 0.05);//Parar com GAP
de 5%
         break;
         case 2:
            model.getEnv().set(GRB_DoubleParam_MIPGap, 0.8);//Parar com GAP de
80%
         break;
      }
      model.getEnv().set(GRB_DoubleParam_Heuristics, 0.05);//Heurística 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_{jk} 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 restrição 3*/

//-------------------------------------------------------------
//restrição 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 restrição 4*/

                //------------------------------------------------------------
                //restrição 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 restrição
5

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

                //------------------------------------------------------------
                //Exporta a solução
                //------------------------------------------------------------
                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 - Código do erro = " << e.getErrorCode() << endl;
		cout << e.getMessage() << endl;
	}catch(...)
	{
		cout << "Gurobi - Erro durante otimização" << 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 heuristicaconstrutiva: 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
grande
                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
tempo
                        posmenortempo= i; //guarda a posição do menor
tempo
                    }

                for (int i=0; i<njob; i++)
                    if (maquina[job2[posmenortempo].maq[k]].tarefa[i]<0)
        //verifica se a posição de memória está vazia
                    {
                        maquina[job2[posmenortempo].maq[k]].tarefa[i]=
posmenortempo;     //máquina recebe a tarefa
                        if(k==0)
                        {
                            if (i==0)

        maquina[job2[posmenortempo].maq[k]].tempoini[i]= 0;          //se for a primeira
tarefa da máquina o tempo inicial é zero
```

```
                                        else

        maquina[job2[posmenortempo].maq[k]].tempoini[i]=
maquina[job2[posmenortempo].maq[k]].tempofim[i-1];          //se não for a primeira
tarefa da máquina o tempo inicial é o tempo final da tarefa anterior

        maquina[job2[posmenortempo].maq[k]].tempofim[i]=
maquina[job2[posmenortempo].maq[k]].tempoini[i] +job2[posmenortempo].tp[k];
                                    }else //k>0
                                    {
                                        if (i==0)//se for a primeira tarefa da máquina
                                        {
                                            for (int l=0; l<njob; l++)
                                                if
(maquina[job2[posmenortempo].maq[k-1]].tarefa[l]== posmenortempo)
                                                    postarefamaqanterior= l;
                                            if
(maquina[job2[posmenortempo].maq[k-1]].tempofim[postarefamaqanterior]>0)

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

        maquina[job2[posmenortempo].maq[k]].tempoini[i]= 0;
                                        }else
                                        {
                                            for (int l=0; l<njob; l++)
                                                if
(maquina[job2[posmenortempo].maq[k-1]].tarefa[l]== posmenortempo)
                                                    postarefamaqanterior= l;
                                            if
(maquina[job2[posmenortempo].maq[k-1]].tempofim[postarefamaqanterior] >
maquina[job2[posmenortempo].maq[k]].tempofim[i-1])
```

```
        maquina[job2[posmenortempo].maq[k]].tempoini[i]=
maquina[job2[posmenortempo].maq[k-1]].tempofim[postarefamaqanterior];
                                                    else

        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
s
```

```
//---------------------------------------------------------------------------
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
(vector job)
					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)) //é primeira tarefa da
máquina e é a primeira máquina 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 máquinas alocadas
                                                    seqtar[i]++;
        //sequencia de tarefas alocadas
                                                }else
                                                {

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

    maquina[job[i].maq[seqtar[i]]].tempoini[seqmaq[job[i].maq[seqtar[i]]]]=
tempofimtar[i];
                                                    else if(seqtar[i]==0)//não é a
primeira tarefa da máquina mas é a primeira máquina da tarefa

    maquina[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] >
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]= INT_MAX;

        maquina[k].tempofim[i]= INT_MAX;
                                                    }
                                                }
                                            }
                    }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)) //é primeira tarefa da
máquina e é 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) //é a primeira tarefa da máquina mas não
é 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 é a
primeira tarefa da máquina mas é 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 é a primeira tarefa da
máquina nem é 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]= INT_MAX;

        maquina1[k].tempofim[i]= INT_MAX;
                                                                }
                                                }
                                        }
                        }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)) //é primeira tarefa da
máquina e é 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) //é a primeira tarefa da máquina mas não é 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 é a primeira tarefa da máquina mas é 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]= INT_MAX;

        maquina2[k].tempofim[i]= INT_MAX;
                                                    }
                                            }
                                        }
                        }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 aleatóriamente 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 aleatóriamente 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 aleatóriamente 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 aleatóriamente 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);
    }
}

//---------------------------------------------------------------------------
//Função troca6opt: faz uma troca 6 optimal na solução s
//---------------------------------------------------------------------------
void troca6opt(int s)
{
        int i, j, maq;
```

```
        //Escolhe aleatóriamente as tarefas a serem trocas
    for(int l=0; l<6; l++)
    {
        i= rand()%(njob-1);
        j= rand()%(njob-1);
        maq= rand()%(nmaq-1);
        troca1opt(i, j, maq, s);
    }
}

//---------------------------------------------------------------------------
//Função buscalocal: faz uma busca local utilizando troca 1 opt
//---------------------------------------------------------------------------
void buscalocal(int s)
{
        int i, j, maq;
        int melhor_fo= INT_MAX;
        switch (s)
        {
                case 0: //ILS
                        FOs3= 2*FOs0;  //inicializa com um número grande
                        maquina.resize(0);          //solução s recebe solução s0
                        for (int l=0; l<nmaq; l++)
                                maquina.push_back(maquina0[l]);
                        FO= FOs0;

                        for (int k=0; k<nmaq; k++)              //para todas as máquinas
(vector maquina)
                                for (int i=0; i<njob-1; i++)        //para todas as tarefas
(vector maquina)
                                        for (int j=i+1; j<njob; j++)   //para todas as tarefas
(vector maquina)
                                        {
```

```
                                        troca1opt(i,j,k,0);
                                        calculatempo(0);
                                        calculafo(0);

                                        if(FO>=FOs0) //se houve piora
                                        {
                                                maquina.resize(0);          //solução s
recebe solução s0
                                                for (int l=0; l<nmaq; l++)

        maquina.push_back(maquina0[l]);
                                                FO= FOs0;
                                        }else{ //se houve melhoria
                                                maquina3.resize(0);         //solução s3
recebe solução s
                                                for (int l=0; l<nmaq; l++)

        maquina3.push_back(maquina[l]);
                                                FOs3= FO;
                                                maquina.resize(0);          //solução s
recebe solução s0
                                                for (int l=0; l<nmaq; l++)

        maquina.push_back(maquina0[l]);
                                                FO= FOs0;
                                        }
                                }
                        if(FOs3<FOs0)
                        {
                                maquina.resize(0);          //solução s recebe solução s3
                                for (int l=0; l<nmaq; l++)
                                        maquina.push_back(maquina3[l]);
                                FO= FOs3;
                        }else{
```

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

            case 1: //perturbação do ILS
                FOs3= 2*FOs1;  //inicializa com um número grande
                maquina2.resize(0);         //solução s2 recebe solução s1
                for (int l=0; l<nmaq; l++)
                    maquina2.push_back(maquina1[l]);
                FOs2= FOs1;

                for (int k=0; k<nmaq; k++)                  //para todas as máquinas
(vector maquina)
                    for (int i=0; i<njob-1; i++)            //para todas as tarefas
(vector maquina)
                        for (int j=i+1; j<njob; j++)    //para todas as tarefas
(vector maquina)
                        {
                            troca1opt(i,j,k,2);
                            calculatempo(2);
                            calculafo(2);

                            if(FOs2>=FOs1) //se houve piora
                            {
                                maquina2.resize(0);         //solução s2
recebe solução s1
                                for (int l=0; l<nmaq; l++)

        maquina2.push_back(maquina1[l]);
                                FOs2= FOs1;
                            }else{ //se houve melhoria
```

```
                                                maquina3.resize(0);          //solução s3
recebe solução s
                                                for (int l=0; l<nmaq; l++)

        maquina3.push_back(maquina2[l]);
                                                FOs3= FOs2;
                                                maquina2.resize(0);          //solução s
recebe solução s0
                                                for (int l=0; l<nmaq; l++)

        maquina2.push_back(maquina1[l]);
                                                FOs2= FOs1;
                                        }
                                }
                if(FOs3<FOs1)
                {
                        maquina1.resize(0);          //solução s recebe solução s3
                        for (int l=0; l<nmaq; l++)
                                maquina1.push_back(maquina3[l]);
                        FOs1= FOs3;
                }else{
                        maquina2.resize(0);          //solução s recebe solução s0
                        for (int l=0; l<nmaq; l++)
                                maquina2.push_back(maquina1[l]);
                        FOs2= FOs1;
                }
        break;

        case 2: //ILS e VNS
                FOs3= 2*FOs1;  //inicializa com um número grande
                maquina2.resize(0);          //solução s2 recebe solução s1
                for (int l=0; l<nmaq; l++)
                        maquina2.push_back(maquina1[l]);
                FOs2= FOs1;
```

```
                for (int k=0; k<nmaq; k++)              //para todas as máquinas
(vector maquina)
                        for (int i=0; i<njob-1; i++)        //para todas as tarefas
(vector maquina)
                                for (int j=i+1; j<njob; j++)  //para todas as tarefas
(vector maquina)
                                {
                                        troca1opt(i,j,k,2);
                                        calculatempo(2);
                                        calculafo(2);

                                        if(FOs2>=FOs1)     //se houve piora
                                        {
                                                maquina2.resize(0);         //solução s2
recebe solução s1
                                                for (int l=0; l<nmaq; l++)

        maquina2.push_back(maquina1[l]);
                                                FOs2= FOs1;
                                        }else{ //se houve melhoria
                                                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;
                }
            break;

            case 3: //busca local da BT
                FOs3= 2*FO;  //inicializa com um número grande
                maquina1.resize(0);         //solução s recebe solução s0
                for (int l=0; l<nmaq; l++)
                    maquina1.push_back(maquina[l]);
                FOs1= FO;

                for (int k=0; k<nmaq; k++)                  //para todas as
máquinas (vector maquina)
                    for (int i=0; i<njob-1; i++)            //para todas as tarefas
(vector maquina)
                        for (int j=i+1; j<njob; j++)    //para todas as tarefas
(vector maquina)
                        {
                            troca1opt(i,j,k,1); //TROCA O OPT AQUI
                            calculatempo(1);
                            calculafo(1);
                            bool estanalistatabu = esta_lista_tabu(i, j, k ,
&iniciotabu);
```

```
                                        if (!estanalistatabu || (estanalistatabu &&
FOs1<FOs0))
                                                if(melhor_fo >= FOs1)
                                                {
                                                        melhor_i= i; //guarda os
elementos TABU
                                                        melhor_j= j;
                                                        melhor_maq= k;
                                                        maquina3.resize(0);
        //solução s3 recebe solução s1
                                                        for (int l=0; l<nmaq; l++)

        maquina3.push_back(maquina1[l]);
                                                        FOs3= FOs1;
                                                        maquina1.resize(0);
        //solução s1 recebe solução s
                                                        for (int l=0; l<nmaq; l++)

        maquina1.push_back(maquina[l]);
                                                        FOs1= FO;
                                                }else{//se houve piora
                                                        maquina1.resize(0);
        //solução s1 recebe solução s
                                                        for (int l=0; l<nmaq; l++)

        maquina1.push_back(maquina[l]);
                                                        FOs1= FO;
                                                }
                                        }

                maquina.resize(0);          //solução s recebe solução s3
                for (int l=0; l<nmaq; l++)
                        maquina.push_back(maquina3[l]);
                FO= FOs3;
```

```
                break;
        }
}

//---------------------------------------------------------------------------
//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<ITERMAXVNS)         //critério de parada: número de iterações
        //while (tempo<TEMPOMAXVNS)                 //critério de parada: tempo
computacional
        {
                k= 1;                   //estrutura de vizinhança atual
                iter++;                 //conta o número de iterações
                time(&t_fim);
                tempo= difftime(t_fim,t_ini);
                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(2);
        if (FOs2<FO)
        {
            maquina.resize(0);          //solução s recebe solução s2
```

```
                    maquina1.resize(0);          //solução s1 recebe solução s2
                    for (int l=0; l<nmaq; l++)
                    {
                        maquina.push_back(maquina2[l]);
                        maquina1.push_back(maquina2[l]);
                    }
                    FO= FOs2;
                    FOs1= FOs2;
                    cout << endl;
                    for(j=0;j<MAX_MAQ;j++)
                    {
                        for(i=0;i<MAX_JOB;i++)
                        {
                        cout <<        maquina[j].tarefa[i] << " ";
                        }
                        cout << endl;
                    }
                    cout << endl;
                    k= 1;
    //iter= 0; //zera o número de iterações sem melhora
                }else
                {
                    k++;
                    maquina1.resize(0);          //solução s1 recebe solução s
                    for (int l=0; l<nmaq; l++)
                        maquina1.push_back(maquina[l]);
                    FOs1= FO;
                }
        }
}

if(FOs0<FO)
{
        maquina.resize(0);          //solução s recebe solução 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 perturbacao: perturba a solução do ils
//---------------------------------------------------------------------------
void perturbacao()
{
        int i, j;

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

        troca2opt(1); //perturbação de nível 1
   calculatempo(1);
        calculafo(1);
        buscalocal(1);
        if(FOs3>FOs1) //se não houve melhoria com a perturbação de nível 1
        {
                troca3opt(1); //perturbação de nível 2
                calculatempo(1);
                calculafo(1);
                buscalocal(1);
                if(FOs3>FOs1) //se não houve melhoria com a perturbação de nível 2
```

```
                {
                        troca4opt(1); //perturbação de nível 3
                        calculatempo(1);
                        calculafo(1);
                        buscalocal(1);
                }
        }
}

//-------------------------------------------------------------------------
//Função ils: Controla a execução da busca local iterativa
//-------------------------------------------------------------------------
void ils()
{
        cout<<"\nIniciando a metaheurística ILS.\n\n";
        time(&t_ini);

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

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

        buscalocal(0);
        while (iter<ITERMAXILS)         //critério de parada: número de iterações
        //while (tempo<TEMPOMAXILS)         //critério de parada: tempo
computacional
        {
                iter++;
                time(&t_fim);
                tempo= difftime(t_fim,t_ini);
                cout<<"\nIteração "<<iter<<" - FO "<<FO;
```

```
                perturbacao();
                //cout<<"\nIteração "<<iter<<" - FO perturbada "<<FOs1;
                buscalocal(2);
                //cout<<"\nIteração "<<iter<<" - FO busca "<<FOs2;
                if (FOs2<FO)
                {
                        maquina.resize(0);          //solução s recebe solução s2
                        for (int l=0; l<nmaq; l++)
                                maquina.push_back(maquina2[l]);
                        FO= FOs2;
                        //iter= 0; //zera o número de iterações sem melhora
                }
        }

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

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


//-------------------------------------------------------------------------
//Função Busca Tabu
//-------------------------------------------------------------------------
void buscatabu()
{
```

```
	cout<<"\nIniciando a metaheurística Busca Tabu.\n\n";
	time(&t_ini);

	int iter= 0;		//número atual de iterações
	int melhoriter= 0;	//número da melhor s* iteração mais recente

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

	iniciotabu= finaltabu= NULL; //inicializa a lista tabu zerada

	while (iter-melhoriter < ITERMAXBT)		//critério de parada: número de iterações
	//while (tempo<TEMPOMAXBT)		//critério de parada: tempo computacional
	{
		iter++;
		time(&t_fim);
		tempo= difftime(t_fim,t_ini);
		buscalocal(3);
		cout<<"\nIteração "<<iter<<" - FO "<<FO;
		tabu= (struct lista *)malloc(sizeof(struct lista));//atualizar a lista tabu
		tabu->i= melhor_i;
		tabu->j= melhor_j;
		tabu->maq= melhor_maq;
		insere_lista_tabu(tabu, &iniciotabu, &finaltabu);
		cout<<"\nA lista TABU está com "<<lenght_lista_tabu(&iniciotabu, &finaltabu)<<" elementos.\n";
		display_lista_tabu(&iniciotabu, &finaltabu);
		if (lenght_lista_tabu(&iniciotabu, &finaltabu) > TAMMAX_LISTA)
			apaga_registro_tabu(iniciotabu, &iniciotabu, &finaltabu); //vou apagar o primeiro registro da lista tabu
```

```
            if (FO<FOs0)
            {
                  maquina0.resize(0);          //solução s0 recebe solução s
                  for (int l=0; l<nmaq; l++)
                        maquina0.push_back(maquina[l]);
                  FOs0= FO;
                  melhoriter= iter; //atualiza a interação da melhor solução
            }else
            {     //retorna para a melhor solução conhecida
                  maquina.resize(0);           //solução s recebe solução s0
                  for (int l=0; l<nmaq; l++)
                        maquina.push_back(maquina0[l]);
                  FO= FOs0;
            }
      }

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

      cout<<"\nIteração "<<iter<<" - FO "<<FOs0;
      cout<<"\nMetaheurística Busca TABU concluída.\n\n";
      time(&t_fim);
      tempo= difftime(t_fim,t_ini);
      cout<<"O problema foi resolvido em "<<tempo<<" segundos.\n";
}

//---------------------------------------------------------------------------
```

```
//Função insere_lista_tabu: Insere o registro i no final de uma lista duplamente
encadeada
//-------------------------------------------------------------------------
void insere_lista_tabu(struct lista *i, struct lista **iniciotabu, struct lista **finaltabu)
{
	struct lista *antigo, *ponteiro;

	if (*finaltabu== NULL)
	{
		i->proximo= NULL;
		i->anterior= NULL;
		*finaltabu= i;
		*iniciotabu= i;
		return;
	}
	ponteiro= *iniciotabu;
	antigo= NULL;
	while (ponteiro)
	{
		antigo= ponteiro;
		ponteiro= ponteiro->proximo;
	}
	antigo->proximo= i;
	i->proximo= NULL;
	i->anterior= antigo;
	*finaltabu= i;
}

//-------------------------------------------------------------------------
//Função lenght_lista_tabu: Mostra o comprimento da lista completa
//-------------------------------------------------------------------------
int lenght_lista_tabu(struct lista **iniciotabu, struct lista **finaltabu)
{
	struct lista *registro= *iniciotabu;
```

```
	int cont= 0;

	while(registro)
	{
		registro= registro->proximo;
		cont++;
	}

	return cont;
}

//---------------------------------------------------------------------------
//Função apaga_registro_tabu: Remove o elemento mais antigo da lista
//---------------------------------------------------------------------------
void apaga_registro_tabu(struct lista *registro, struct lista **iniciotabu, struct lista
**finaltabu)
{
	if(registro)
	{
		if(*iniciotabu== registro)
		{
			*iniciotabu= registro->proximo;
			if(*iniciotabu)
				(*iniciotabu)->anterior = NULL;
			else
				*finaltabu = NULL;
		}
		else
		{
			registro->anterior->proximo= registro->proximo;
			if(registro!= *finaltabu)
				registro->proximo->anterior= registro->anterior;
			else
				*finaltabu= registro->anterior;
```

```
                }
                free(registro);
        }
}

//-------------------------------------------------------------------------
//Função display_lista_tabu: Mostra o conteúdo da lista completa
//-------------------------------------------------------------------------
void display_lista_tabu(struct lista **iniciotabu, struct lista **finaltabu)
{
        struct lista *registro = *iniciotabu;
        int cont = 0;

        while(registro)
        {
                cout<<"Lista Tabu["<<cont<<"] --> i= "<<registro->i<<"; j= "<<registro-
>j<<"; maq= "<<registro->maq<<";\n";
                registro = registro->proximo;
                cont++;
        }
}

//-------------------------------------------------------------------------
//Função esta_lista_tabu: Verifica se a posição está na lista tabu
//-------------------------------------------------------------------------
bool esta_lista_tabu(int i, int j, int maq, struct lista **iniciotabu)
{
        struct lista *registro = *iniciotabu;

        while(registro)
        {
                if ((registro->i == i) && (registro->j == j) && (registro->maq == maq))
                        return true;
                registro = registro->proximo;  //obtém novo registro
```

```
        }
        return false; //não encontrou
}

//---------------------------------------------------------------------------
//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";
}

//---------------------------------------------------------------------------
//Função metaheurística híbrida VNS e Busca Tabu
//---------------------------------------------------------------------------
void vnsbuscatabu()
{
        cout<<"\nIniciando a metaheurística híbrida VNS e Busca Tabu.\n\n";
        time(&t_ini);

        int iter= 0;            //número atual de iterações
        int melhoriter= 0;      //número da melhor s* iteração mais recente
        int i, j, maq;
        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;

        iniciotabu= finaltabu= NULL; //inicializa a lista tabu zerada

        while (iter-melhoriter < ITERMAXVNS)          //critério de parada: número de
iterações
        //while (tempo<TEMPOMAXVNS)                   //critério de parada: tempo
computacional
        {
                k= 1;                   //estrutura de vizinhança atual
                iter++;                 //conta o número de iterações
                time(&t_fim);
```

```
tempo= difftime(t_fim,t_ini);
while (k<= KMAX)
{
        switch (k)
        {
                case 1:
                        i= rand()%(njob-1);
                        j= rand()%(njob-1);
                        maq= rand()%(nmaq-1);
                        troca1opt(i, j, maq, 0);
                        calculatempo(0);
                        calculafo(0);
                break;

                case 2:
                        troca2opt(0);
                        calculatempo(0);
                        calculafo(0);
                break;

                case 3:
                        troca3opt(0);
                        calculatempo(0);
                        calculafo(0);
                break;

                case 4:
                        troca4opt(0);
                        calculatempo(0);
                        calculafo(0);
                break;

                case 5:
                        troca5opt(0);
```

```
                    calculatempo(0);
                    calculafo(0);
                break;

                case 6:
                    troca6opt(0);
                    calculatempo(0);
                    calculafo(0);
                break;
            }

            buscalocal(3);
            cout<<"\nIteração "<<iter<<" - FO "<<FO;
            tabu= (struct lista *)malloc(sizeof(struct lista));//atualizar a lista
tabu
            tabu->i= melhor_i;
            tabu->j= melhor_j;
            tabu->maq= melhor_maq;
            insere_lista_tabu(tabu, &iniciotabu, &finaltabu);
            cout<<"\nA lista TABU está com
"<<lenght_lista_tabu(&iniciotabu, &finaltabu)<<" elementos.\n";
            display_lista_tabu(&iniciotabu, &finaltabu);
            if (lenght_lista_tabu(&iniciotabu, &finaltabu) > TAMMAX_LISTA)
                apaga_registro_tabu(iniciotabu, &iniciotabu, &finaltabu);
//vou apagar o primeiro registro da lista tabu

            if (FO<FOs0)
            {
                maquina0.resize(0);          //solução s0 recebe solução s
                for (int l=0; l<nmaq; l++)
                    maquina0.push_back(maquina[l]);
                FOs0= FO;
                melhoriter= iter; //atualiza a interação da melhor solução
```

```
                        k= 1;   //retorna à primeira estrutura de vizinhança em
caso de melhoria
                    }else
                    {
                        k++; //avança para a próxima estrutura de vizinhança
                        maquina.resize(0);          //solução s1 recebe solução s
                        for (int l=0; l<nmaq; l++)
                            maquina.push_back(maquina0[l]);
                        FO= FOs0;
                    }
                }
            }

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

            cout<<"\nIteração "<<iter<<" - FO "<<FOs0;
            cout<<"\nMetaheurística híbrida VNS e Busca Tabu concluída.\n\n";
            time(&t_fim);
            tempo= difftime(t_fim,t_ini);
            cout<<"O problema foi resolvido em "<<tempo<<" segundos.\n";
}

//---------------------------------------------------------------------------
//Função Simulated Annealing
//---------------------------------------------------------------------------
void simannealing()
{
            float delta,x;
```

```
    int iterT=0,vizinho=1,i,j,maq,contador=0,retido=5;
    double Temp = temperatura;
    cout<<"\nIniciando a metaheurística Simulated Annealing.\n\n";
    time(&t_ini);
    maquina0.resize(0);          //solução s0 -- s* recebe solução s
    for (int l=0; l<nmaq; l++)
        maquina0.push_back(maquina[l]);
    FOs0= FO;
maquina1.resize(0);              //solução s1 -- s' recebe solução s
    for (int l=0; l<nmaq; l++)
        maquina1.push_back(maquina[l]);
    FOs1= FO;
    while (Temp>0.1)
    {
        while (iterT < SAMax)
        {
            iterT = iterT + 1;
            switch (vizinho)
            {
                case 1:
                    i= rand()%(njob-1);
                    j= rand()%(njob-1);
                    maq= rand()%(nmaq-1);
                    troca1opt(i, j, maq, 1);
                    calculatempo(1);
                    calculafo(1);
                    contador=contador+1;
                    if(contador==retido)
                    {
                        vizinho=vizinho+1;
                        contador=0;
                    }
                break;
```

```
case 2:
        troca2opt(1);
        calculatempo(1);
        calculafo(1);
        contador=contador+1;
        if(contador==retido)
        {
                vizinho=vizinho+1;
                contador=0;
        }
break;

case 3:
        troca3opt(1);
        calculatempo(1);
        calculafo(1);
        contador=contador+1;
        if(contador==retido)
        {
                vizinho=vizinho+1;
                contador=0;
        }
break;
case 4:
        troca4opt(1);
        calculatempo(1);
        calculafo(1);
        contador=contador+1;
        if(contador==retido)
        {
                vizinho=vizinho+1;
                contador=0;
        }
break;
```

```
                case 5:
                        troca5opt(1);
                        calculatempo(1);
                        calculafo(1);
                        contador=contador+1;
                        if(contador==retido)
                        {
                                vizinho=vizinho+1;
                                contador=0;
                        }
                break;

                case 6:
                        troca6opt(1);
                        calculatempo(1);
                        calculafo(1);
                        contador=contador+1;
                        if(contador==retido+10)
                        {
                                vizinho=1;
                                contador=0;
                        }
                break;
        }
        calculatempo(0);
        calculafo(0);
        delta = FOs1-FO;
        if (delta < 0)
        {
                maquina.resize(0);          //solução s recebe solução s1 --
s'
                for (int l=0; l<nmaq; l++)
                        maquina.push_back(maquina1[l]);
```

```
                        FO= FOs1;
                        if (FOs1<FOs0)
                        {
                                maquina0.resize(0);          //solução s0 -- s* recebe
s1 -- s'
                                for (int l=0; l<nmaq; l++)
                                        maquina0.push_back(maquina1[l]);
                                FOs0= FOs1;
                        }
                        else
                        {
                                x =(rand()%1000);
                                if ((x/1000)<exp((-delta/Temp)))
                                {
                                        maquina.resize(0);           //solução s recebe
s1 -- s'
                                        for (int l=0; l<nmaq; l++)
                                                maquina.push_back(maquina1[l]);
                                        FO= FOs1;
                                }
                        }
                }
        }
        Temp = alfa*Temp;
        cout << "Temperatura: "<<Temp<<", FO: "<<  FO << endl;
        iterT=0;
    }
        maquina.resize(0);           //solução s recebe solução s* -- s0
        for (int l=0; l<nmaq; l++)
                maquina.push_back(maquina0[l]);
        FO= FOs0;
    calculatempo(0);
    calculafo(0);
    cout << "FO = "<<FO;
```

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

//------------------------------------------------------------------------
//Função respostapseudoaleatoria: Essa função cria uma resposta pseudoaleatória baseada
//na ordem em que os jobs podem ser posicionados nas máquinas
//------------------------------------------------------------------------
void respostapseudoaleatoria(int s,int x)
{
	int i,j,k,l,contador1=0,contador2,apagador,n;
	vector<int>vetorsorteio; //Vetor utilizado para sortear a ordem dos jobs nas máquinas
	vector<int>::iterator it;
	for (i=0;i<MAX_MAQ;i++)
	{
		for (j=0;j<MAX_JOB;j++)
		{
			maquina[i].tarefa[j] = -1;
		}
	} //Torna todos os elementos da matriz "-1"
	for (k=0;k<MAX_MAQ;k++)
	{			//Esse conjunto de 3 laços verifica todas as máquinas em todas as colunas
		for (i=0;i<MAX_MAQ;i++)
		{
			vetorsorteio.resize(0);
			for (j=0;j<MAX_JOB;j++)
			{
				if (job[j].maq[k]==i) //Caso o elemento seja igual a máquina i adiciona o número do job no vetor
```

```
                        {
                         vetorsorteio.push_back(j);
                         contador1=contador1+1;  //Variável contadora que
constata que existe um vetor a ser sorteado
                        }
                }
                if((contador1!=0) && (vetorsorteio.size()>1) ) //Caso esse vetor
exista e seu valor seja maior que 1, entra no laço
                {
                        while(vetorsorteio.size()>1) //Roda enquanto o tamanho
do vetor for maior que 1, pois caso fosse 0 daria erro na função rand()
                        {
                                contador2=0;
                                n = rand()%(vetorsorteio.size()-1); //Sorteia o
número
                                for (l=0;l<MAX_JOB;l++)
                                {
                                        if(maquina[i].tarefa[l]==-1)
                                        {
                                                contador2=contador2+1;  //Conta a
quantidade de -1 existentes na linha da máquina i em questão
                                        }
                                }
                                maquina[i].tarefa[(MAX_JOB-
contador2)]=vetorsorteio[n]; //Adiciona a tarefa na posição correta em função do
número de -1 existentes na linha
                                for(l=0;l<vetorsorteio.size();l++)
                                {

        if(vetorsorteio[l]==maquina[i].tarefa[MAX_JOB-contador2])
                                        {
                                                apagador = l; //Captura a posição do
elemento que foi sorteado e portanto deverá ser apagado do vetor
                                        }
```

```
                    }
                    it = vetorsorteio.begin()+apagador;
                    vetorsorteio.erase(it); //Apaga o elemento do vetor
                }
                contador2=0;
                for (l=0;l<MAX_JOB;l++) //Realiza o mesmo procedimento anterior mas apenas para o último elemento do vetor sorteio
                    {
                        if(maquina[i].tarefa[l]==-1)
                        {
                            contador2=contador2+1;
                        }
                    }
                maquina[i].tarefa[(MAX_JOB-contador2)]=vetorsorteio[0];
                it = vetorsorteio.begin();
                vetorsorteio.erase(it);
            }
            if(contador1!=0 && (vetorsorteio.size()==1)) //Entra nessa condição caso somente tenha um elemento no vetor sorteio inicialmente e faz o mesmo procedimento
            {
                contador2=0;
                for (l=0;l<MAX_JOB;l++)
                    {
                        if(maquina[i].tarefa[l]==-1)
                        {
                            contador2=contador2+1;
                        }
                    }
                maquina[i].tarefa[MAX_JOB-contador2]=vetorsorteio[0];
            }
            contador1=0;
            contador2=0;
        }
```

```
        }
        for(j=0;j<MAX_MAQ;j++)
        {
                for(i=0;i<MAX_JOB;i++)
                {
                        cout <<         maquina[j].tarefa[i] << " ";
                }
                cout << endl;
        }
        calculatempo(0);
        calculafo(0);
}

//---------------------------------------------------------------------------
//Função respostaordenatempo: Cria uma resposta baseada no tempo total de
execução
//de cada job. Os jobs que necessitam de menos tempo total para serem realizados
//são selecionados primeiro na resposta
//---------------------------------------------------------------------------
void respostaordenatempo()
{
        vector<double>somatempojobs;
        vector<double>vetorposicaojobs;

        int soma=0,maxi=0,troca1,troca2,i,j,troca;
        for (int i=0; i<njob; i++)          //Faz a soma dos tempos totais dos jobs
        {
                for (int k=0; k<nmaq; k++)
                {
                        soma = soma + job[i].tp[k];
                }
                somatempojobs.push_back(soma);
                soma = 0;
        }
```

```
        for (int i=0; i<njob; i++)        //Adiciona os jobs em ordem em um segundo
vetor
        {
                vetorposicaojobs.push_back(i);
        }
        for (j=0;j<somatempojobs.size()-1;j++) //Ordena o vetor com os tempos totais
dos jobs juntamente com o vetor números dos jobs
   {
     for (i=0;i<=somatempojobs.size()-j-2;i++)
     {
       if(somatempojobs[i]>somatempojobs[i+1])
       {
                        troca1=somatempojobs[i+1];
                        somatempojobs[i+1]=somatempojobs[i];
                        somatempojobs[i]=troca1;
                        troca2=vetorposicaojobs[i+1];
                        vetorposicaojobs[i+1]=vetorposicaojobs[i];
                        vetorposicaojobs[i]=troca2;
                }
     }
   }
   for (i=0;i<njob;i++)   //Faz a coluna de resposta ser igual aos jobs ordenados por
tempo total de utilização
        {
                for (j=0;j<nmaq;j++)
                {
                        maquina[j].tarefa[i] = vetorposicaojobs[i];
                }
        }
        for (i=0;i<nmaq;i++)
        {
                for (j=0;j<njob;j++)
                {
                        cout << maquina[i].tarefa[j] << " ";
```

```
                }
                cout << endl;
        }
        cout << endl;
         calculatempo(0);
         calculafo(0);
         cout << FO << endl;

}

//---------------------------------------------------------------------------
//Função menorrespostaaleatoria: Essa função cria uma resposta pseudoaleatória baseada
//na ordem em que os jobs podem ser posicionados nas máquinas repetidas vezes e seleciona
//a menor FO dentre todas as respostas geradas
//---------------------------------------------------------------------------
void menorrespostaaleatoria(int s,int x)
{
        int i,j,k,l,contador1=0,contador2,apagador,n,o;
        vector<int>vetorsorteio; //Vetor utilizado para sortear a ordem dos jobs nas máquinas
        vector<int>::iterator it;
        FOs1=500000;
        for(o=0;o<10000;o++)
        {
                for (i=0;i<MAX_MAQ;i++)
                {
                        for (j=0;j<MAX_JOB;j++)
                        {
                                maquina[i].tarefa[j] = -1;
                        }
                } //Torna todos os elementos da matriz "-1"
                for (k=0;k<MAX_MAQ;k++)
```

```
        {                       //Esse conjunto de 3 laços verifica todas as
máquinas em todas as colunas
                for (i=0;i<MAX_MAQ;i++)
                {
                        vetorsorteio.resize(0);
                        for (j=0;j<MAX_JOB;j++)
                        {
                                if (job[j].maq[k]==i) //Caso o elemento seja igual a
máquina i adiciona o número do job no vetor
                                {
                                 vetorsorteio.push_back(j);
                                 contador1=contador1+1;  //Variável contadora que
constata que existe um vetor a ser sorteado
                                }
                        }
                        if((contador1!=0) && (vetorsorteio.size()>1) ) //Caso esse
vetor exista e seu valor seja maior que 1, entra no laço
                        {
                                while(vetorsorteio.size()>1) //Roda enquanto o
tamanho do vetor for maior que 1, pois caso fosse 0 daria erro na função rand()
                                {
                                        contador2=0;
                                        n = rand()%(vetorsorteio.size()-1); //Sorteia o
número
                                        for (l=0;l<MAX_JOB;l++)
                                        {
                                                if(maquina[i].tarefa[l]==-1)
                                                {
                                                        contador2=contador2+1;
//Conta a quantidade de -1 existentes na linha da máquina i em questão
                                                }
                                        }
```

```
                                        maquina[i].tarefa[(MAX_JOB-
contador2)]=vetorsorteio[n]; //Adiciona a tarefa na posição correta em função do
número de -1 existentes na linha
                                        for(l=0;l<vetorsorteio.size();l++)
                                        {

        if(vetorsorteio[l]==maquina[i].tarefa[MAX_JOB-contador2])
                                                {
                                                        apagador = l; //Captura a
posição do elemento que foi sorteado e portanto deverá ser apagado do vetor
                                                }
                                        }
                                        it = vetorsorteio.begin()+apagador;
                                        vetorsorteio.erase(it); //Apaga o elemento do
vetor
                                }
                                contador2=0;
                                for (l=0;l<MAX_JOB;l++) //Realiza o mesmo
procedimento anterior mas apenas para o último elemento do vetor sorteio
                                        {
                                                if(maquina[i].tarefa[l]==-1)
                                                {
                                                        contador2=contador2+1;
                                                }
                                        }
                                maquina[i].tarefa[(MAX_JOB-
contador2)]=vetorsorteio[0];
                                it = vetorsorteio.begin();
                                vetorsorteio.erase(it);
                        }
                        if(contador1!=0 && (vetorsorteio.size()==1)) //Entra nessa
condição caso somente tenha um elemento no vetor sorteio inicialmente e faz o
mesmo procedimento
                        {
```

```
                                        contador2=0;
                                        for (l=0;l<MAX_JOB;l++)
                                                {
                                                        if(maquina[i].tarefa[l]==-1)
                                                        {
                                                                contador2=contador2+1;
                                                        }
                                                }
                                        maquina[i].tarefa[MAX_JOB-
contador2]=vetorsorteio[0];
                                }
                                contador1=0;
                                contador2=0;
                        }
                }
                for(j=0;j<MAX_MAQ;j++)
                {
                        for(i=0;i<MAX_JOB;i++)
                        {
                                cout <<         maquina[j].tarefa[i] << " ";
                        }
                        cout << endl;
                }
                calculatempo(0);
                calculafo(0);
                if(FO<FOs1)
                {
                        maquina1.resize(0);          //solução s1 recebe solução s
                        for (int l=0; l<nmaq; l++)
                                maquina1.push_back(maquina[l]);
                        FOs1= FO;
                }
        }
        maquina.resize(0);           //solução s recebe solução s1
```

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

void heuristicaderefinamento()
{
        int naomelhoria,linha,i,j,k,troca1,troca2,troca3,iteracao=1,l=0,w=0,cond;
        vector<int>jobportempo;
        vector<int>jobdisponivel;
        vector<int>ultimamaq;
        vector<int>tempodisp;
        vector<int>posicao;
        vector<int>linhafinal;
        vector<int>::iterator it;
   //respostaordenatempo();
   //heuristicaconstrutiva();
   //respostapseudoaleatoria(MAX_MAQ,MAX_JOB);
   //menorrespostaaleatoria(MAX_MAQ,MAX_JOB);
        linha=0;
        naomelhoria=0;
        maquina1.resize(0);          //solução s1 recebe solução s
        for (int l=0; l<nmaq; l++)
                maquina1.push_back(maquina[l]);
        FOs1= FO;
        while (naomelhoria<(nmaq-1))
        {
                maquina.resize(0);           //solução s recebe solução s1
                for (int l=0; l<nmaq; l++)
                        maquina.push_back(maquina1[l]);
                FO= FOs1;
                for(i=0;i<njob;i++)   //Separa os jobs já disponíveis para a máquina
                {
                        if(job[i].maq[0]==linha)
```

```
                {
                        jobdisponivel.push_back(i);
                }
        }
        cout << endl;
        for(i=0;i<njob;i++)  //Cria um vetor com todos os jobs
        {
                jobportempo.push_back(i);
        }
        if(jobdisponivel.size()>0)     //Sinaliza no vetor de tempo os trabalhos já selecionados anteriormente
        {
                for(i=0;i<jobdisponivel.size();i++)
                {
                        for(j=0;j<jobportempo.size();j++)
                        {
                                if(jobportempo[j]==jobdisponivel[i])
                                {
                                        jobportempo[j]=-100;
                                }
                        }
                }
        }
        for(i=0;i<jobportempo.size();i++) //Cria um vetor com o número da máquina anterior à máquina analisada
        {
                for(j=0;j<nmaq;j++)
                {
                        if (job[jobportempo[i]].maq[j]==linha)
                        {
                                if (job[i].maq[j-1]<nmaq)
                                {
                                        ultimamaq.push_back(job[i].maq[j-1]);
                                        k=k+1;
```

```
                        }
                    }
                }
                if(k==0)   //Caso não entre no laço por ser -100, adiciona -100 no outro vetor também para manter a indexação
                {
                    ultimamaq.push_back(-100);
                }
                k=0;
            }
            k=0;
            if(jobportempo.size()>0)
            {
                for(i=0;i<jobportempo.size();i++) //Captura o tempo de finalização na máquina anterior à analisada
                {
                    for(j=0;j<njob;j++)
                    {
                        if(jobportempo[i]==maquina[ultimamaq[i]].tarefa[j])
                        {

    tempodisp.push_back(maquina[ultimamaq[i]].tempofim[j]);
                            k=k+1;
                        }
                    }
                    if(k==0)    //Caso não entre no laço por ser -100, adiciona -100 no outro vetor também para manter a indexação
                    {
                        tempodisp.push_back(-100);
                    }
                    k=0;
                }
            }
            k=0;
```

```
for (j=0;j<jobportempo.size()-1;j++) //Coloca os jobs já selecionados marcados como -100 para o final do vetor
{
	for (i=0;i<=jobportempo.size()-j-2;i++)
	{
		if(jobportempo[i]==(-100))
			{
				troca1=jobportempo[i+1];
				jobportempo[i+1]=jobportempo[i];
				jobportempo[i]=troca1;
				troca2=ultimamaq[i+1];
				ultimamaq[i+1]=ultimamaq[i];
				ultimamaq[i]=troca2;
				troca3=tempodisp[i+1];
				tempodisp[i+1]=tempodisp[i];
				tempodisp[i]=troca3;
			}
	}
}
i=jobportempo[jobportempo.size()-1];
while(i==-100)   //Exclui os jobs não selecionados do vetor
{
	it = jobportempo.begin()+(jobportempo.size()-1);
	jobportempo.erase(it); //Apaga o elemento do vetor
	it = ultimamaq.begin()+(ultimamaq.size()-1);
	ultimamaq.erase(it); //Apaga o elemento do vetor
	it = tempodisp.begin()+(tempodisp.size()-1);
	tempodisp.erase(it); //Apaga o elemento do vetor
	i=jobportempo[jobportempo.size()-1];
}
if(tempodisp.size()>0)
{
	for (j=0;j<jobportempo.size()-1;j++)//Ordena todos os vetores utilizando como base o tempo de finalização na máquina anterior
```

```
                {
                        for (i=0;i<=jobportempo.size()-j-2;i++)
                        {
                                if(tempodisp[i]>tempodisp[i+1])
                                        {
                                                troca1=jobportempo[i+1];
                                                jobportempo[i+1]=jobportempo[i];
                                                jobportempo[i]=troca1;
                                                troca2=ultimamaq[i+1];
                                                ultimamaq[i+1]=ultimamaq[i];
                                                ultimamaq[i]=troca2;
                                                troca3=tempodisp[i+1];
                                                tempodisp[i+1]=tempodisp[i];
                                                tempodisp[i]=troca3;
                                        }
                        }
                }
                i=0;
                l=0;
                cout << "VIM ATE AQUI" << endl;
                //ERRO DE SEGMENTAÇAO ESTA AQUI
                cond=tempodisp[0];
                tempodisp[0]=0;
                while (tempodisp[i]<=((njob*nmaq)/5)) //Classifica como job disponível
os jobs que não atendem ao critério de tempo mínimo
                {
                        if(i==0)
                        {
                                tempodisp[0]=cond;
                        }
                        if (tempodisp[i]<=((nmaq*njob)))
                        {
                                if(jobdisponivel.size()<=njob)
```

```
                {
                        jobdisponivel.push_back(jobportempo[i]);
                        jobportempo[i]=-100;
                        ultimamaq[i]=-100;
                        tempodisp[i]=-100;
                }
        }
        i=i+1;
}
if(tempodisp.size()>0)
{
        for (j=0;j<jobportempo.size()-1;j++)  //Coloca os jobs já
selecionados marcados como -100 para o final do vetor
        {
                for (i=0;i<=jobportempo.size()-j-2;i++)
                {
                        if(jobportempo[i]==(-100))
                                {
                                        troca1=jobportempo[i+1];
                                        jobportempo[i+1]=jobportempo[i];
                                        jobportempo[i]=troca1;
                                        troca2=ultimamaq[i+1];
                                        ultimamaq[i+1]=ultimamaq[i];
                                        ultimamaq[i]=troca2;
                                        troca3=tempodisp[i+1];
                                        tempodisp[i+1]=tempodisp[i];
                                        tempodisp[i]=troca3;
                                }
                }
        }
}
        i=jobportempo[jobportempo.size()-1];
while(i==-100) //Exclui os jobs não selecionados do vetor
{
```

```
                it = jobportempo.begin()+(jobportempo.size()-1);
                jobportempo.erase(it); //Apaga o elemento do vetor
                it = ultimamaq.begin()+(ultimamaq.size()-1);
                ultimamaq.erase(it); //Apaga o elemento do vetor
                it = tempodisp.begin()+(tempodisp.size()-1);
                tempodisp.erase(it); //Apaga o elemento do vetor
                i=jobportempo[jobportempo.size()-1];
        }
        cout << endl;
        /*for(j=0;j<jobportempo.size();j++)
        {
                cout << jobportempo[j] << " ";
        }*/

        for(i=0;i<jobdisponivel.size();i++) //Classifica a posição dos jobs disponíveis na próxima máquina
        {
                for(j=0;j<nmaq;j++)
                {
                        if (job[jobdisponivel[i]].maq[j]==linha)
                        {
                                posicao.push_back(j+1);
                        }
                }
        }
        if(posicao.size()!=0)
        {
                for (j=0;j<posicao.size()-1;j++) //Ordena os jobs disponíveis baseado na posição deles na próxima máquina
                {
                        for (i=0;i<=posicao.size()-j-2;i++)
                        {
                                if(posicao[i]>posicao[i+1])
                                        {
```

```
                                        troca1=posicao[i+1];
                                        posicao[i+1]=posicao[i];
                                        posicao[i]=troca1;
                                        troca2=jobdisponivel[i+1];
                                        jobdisponivel[i+1]=jobdisponivel[i];
                                        jobdisponivel[i]=troca2;

                                }
                        }
                }
        }
        if(jobdisponivel.size()>0) //Caso tenha jobs disponíveis adiciona eles na
linha output
        {
                for(i=0;i<jobdisponivel.size();i++)
                {
                        linhafinal.push_back(jobdisponivel[i]);
                }
        }
        for(i=0;i<jobportempo.size();i++) //Adiciona os jobs classificados por
tempo na linha output
        {
                linhafinal.push_back(jobportempo[i]);
        }
        for (int i=0; i<njob; i++)          //Transfere a linha output para a matriz
final de resposta
        {
                maquina[linha].tarefa[i]=linhafinal[i];
        }
        calculatempo(0); //Calcula os tempos da nova resposta
        calculafo(0); //Calcula a FO da nova resposta
        if (FO<FOs1)
        {
                maquina1.resize(0);          //solução s1 recebe solução s
```

```
                for (int l=0; l<nmaq; l++)
                        maquina1.push_back(maquina[l]);
                FOs1= FO;
                naomelhoria=0;
        }
        else
        {
                maquina.resize(0);              //solução s recebe solução s1
                for (int l=0; l<nmaq; l++)
                maquina.push_back(maquina1[l]);
                FO= FOs1;
                naomelhoria=naomelhoria+1;
        }
        cout << "Linha da iteração: " << iteracao << endl;
        iteracao=iteracao+1;
        for (int i=0; i<njob; i++)
        {
                cout << maquina[linha].tarefa[i]<< " ";
        }
        linha=linha+1;
        cout << endl;
        if(linha==nmaq)
        {
                linha=0;
        }
        jobportempo.resize(0); //Reinicia o vetor
    jobdisponivel.resize(0); //Reinicia o vetor
    ultimamaq.resize(0); //Reinicia o vetor
    tempodisp.resize(0); //Reinicia o vetor
    posicao.resize(0); //Reinicia o vetor
    linhafinal.resize(0); //Reinicia o vetor
        cout << "Não melhoria = " << naomelhoria << endl;
        cout<< "FO final: " << FOs1 << endl;
}
```

```
}
//---------------------------------------------------------------------------
//Função vns: Controla a execução da busca em vizinhança variável
//---------------------------------------------------------------------------
void vnsmodificado()
{
	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<ITERMAXVNS)	//critério de parada: número de iterações
	//while (tempo<TEMPOMAXVNS)		//critério de parada: tempo
computacional
	{
		k= 1;			//estrutura de vizinhança atual
		iter++;			//conta o número de iterações
		time(&t_fim);
		tempo= difftime(t_fim,t_ini);
		cout<<"\nIteração "<<iter<<" - FO "<<FO;
		while (k<= KMAXMODIFICADO)
		{
			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;
        case 7:
                heuristicaderefinamento();
                calculatempo(1);
                calculafo(1);
        break;
}
buscalocal(2);
if (FOs2<FO)
{
        maquina.resize(0);          //solução s recebe solução s2
        maquina1.resize(0);         //solução s1 recebe solução s2
        for (int l=0; l<nmaq; l++)
        {
                maquina.push_back(maquina2[l]);
                maquina1.push_back(maquina2[l]);
        }
        FO= FOs2;
        FOs1= FOs2;
        cout << endl;
        for(j=0;j<MAX_MAQ;j++)
        {
                for(i=0;i<MAX_JOB;i++)
                {
                cout <<         maquina[j].tarefa[i] << " ";
                }
                cout << endl;
        }
        cout << endl;
        k= 1;
//iter= 0; //zera o número de iterações sem melhora
```

```
                }else
                {
                        k++;
                        maquina1.resize(0);          //solução s1 recebe solução s
                        for (int l=0; l<nmaq; l++)
                                maquina1.push_back(maquina[l]);
                        FOs1= FO;
                }
        }
}

if(FOs0<FO)
{
        maquina.resize(0);          //solução s recebe solução 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";
}
```