UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
COLEGIADO DO CURSO DE ENGENHARIA
DE CONTROLE E AUTOMAÇÃO - CECAU

EDUARDO DE OLIVEIRA FERREIRA

O USO DE TÉCNICAS DE BUSCA EM VIZINHANÇA DE GRANDE PORTE PARA RESOLVER O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS E UNIFORMES

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Ouro Preto, 2018

EDUARDO DE OLIVEIRA FERREIRA

O USO DE TÉCNICAS DE BUSCA EM VIZINHANÇA DE GRANDE PORTE PARA RESOLVER O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS E UNIFORMES

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Prof. Dr. Gustavo Peixoto Silva

Ouro Preto
Escola de Minas – UFOP
2018

F383u Ferreira, Eduardo de Oliveira.
O uso de técnicas de busca em vizinhança de grande porte para resolver o problema de sequenciamento de tarefas em máquinas paralelas e uniformes [manuscrito] / Eduardo de Oliveira Ferreira. - 2018.

55f.:

Orientador: Prof. Dr. Gustavo Peixoto Silva.

Monografia (Graduação). Universidade Federal de Ouro Preto. Escola de Minas. Departamento de Engenharia de Controle e Automação e Técnicas Fundamentais.

1. Sequenciamento de tarefas em ma?quinas paralelas. 2. Busca em vizinhanc?a de grande porte. 3. Dynasearch. 4. Programac?a?o dina?mica. I. Silva, Gustavo Peixoto. II. Universidade Federal de Ouro Preto. III. Titulo.

CDU: 681.5

Catalogação: ficha.sisbin@ufop.edu.br

Monografia defendida e aprovada, em 22 de novembro de 2018, pela comissão avaliadora constituída pelos professores:

Prof. Dr. Gustavo Peixoto Silva - Orientador

Prof. Dr. Agnaldo José da Rocha Reis – Professor Convidado

Prof. M. Sc. Rodolfo Ayala Lopes Costa – Professor Convidado

RESUMO

Este trabalho trata do problema de otimização do sequenciamento em máquinas uniformes e paralelas com atraso total ponderado, conhecido como *Parallel Machines Total Weighted Tardiness Problem*. Para cada tarefa é conhecido o seu tempo de processamento, sua data de entrega e o peso por dia de atraso da conclusão da tarefa em relação à sua data de entrega. Deve-se sequenciar as tarefas entre as máquinas de forma que cada tarefa seja realizada em uma única máquina e cada máquina realize uma única tarefa por vez e sem preempção, com o objetivo de minimizar os atrasos ponderados. Este sequenciamento é obtido em duas etapas: o particionamento das tarefas entre as máquinas e o sequenciamento das tarefas em cada máquina. A contribuição deste trabalho é resolver as duas etapas com heurísticas de busca em vizinhança de grande porte. A técnica *Very Large-scale Neighborhood Search* é empregada de formas distintas daquelas encontradas na literatura para realizar o particionamento das tarefas, e um algoritmo de Programação Dinâmica *Dynasearch* realiza o sequenciamento das tarefas em cada máquina. Ambas as buscas são combinadas na metaheurística *Iterated Local Search* (ILS). Foram realizados testes com problemas *benchmark* da literatura mostrando a competitividade das versões propostas.

Palavras-chave: Sequenciamento de tarefas em máquinas paralelas. Busca em vizinhança de grande porte. *Dynasearch*. Programação dinâmica

ABSTRACT

This work is a study of the Parallel Machines Total Weighted Tardiness Problem. For each given job its processing time, due date and weights are known. The jobs must be allocated in the machines so that each job is executed only once, by one machine only, and each machine performs a single task at a time without preemption, with goal of minimizing the total weighted delays. The scheduling is done in two steps: the partitioning of the tasks between the machines and the sequencing of the jobs inside each machine. The contribution of this work is to solve these two steps using large-scale neighborhood search heuristics. The Very Large-scale Neighborhood Search technique is implemented in ways that are different from those found in the literature to perform the partitioning of tasks, and the Dynasearch Dynamic Programming algorithm performs the job sequencing inside each machine. Both searches are combined in the Iterated Local Search (ILS) metaheuritic. Test results were compared with literature's benchmark problems showing the competitiveness of the proposed techniques.

Keywords: Parallel Machines Scheduling Problem. Very Large-scale Neighborhood Search. *Dynasearch.* Dynamic Programming

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Neste trabalho é abordado o problema de otimização do sequenciamento de uma série de tarefas em um conjunto de máquinas idênticas e que operam em paralelo, conhecido na literatura como *Parallel Machine Total Weighted Tardiness Problem*. Deve-se sequenciar as tarefas entre as máquinas de forma que cada tarefa seja realizada em uma única máquina e cada máquina realize uma única tarefa por vez, com o objetivo de minimizar os atrasos ponderados.

O sequenciamento de tarefas em máquinas paralelas é um problema comum de planejamento. Nele as tarefas devem ser inicialmente particionadas entre as máquinas e posteriormente sequenciadas dentro de cada uma delas. Soluções para o problema determinístico de sequenciamento de tarefas em máquinas paralelas vêm sendo amplamente estudado desde a década de 1950 (GRAHAM *et al.,* 1979). Desde então, muitas sugestões de algoritmos para a solução dessa classe de problemas vêm sendo propostas (AHUJA *et al.,* 2002).

1.1 Descrição do problema

Dado um conjunto de tarefas, cada uma com uma data de entrega, um tempo de processamento e uma penalidade por unidade de tempo de atraso, a serem realizadas sequencialmente por uma única máquina, o problema do sequenciamento de tarefas em uma única máquina (*Single Machine Total Weighted Tardiness Problem*) busca ordenar essas tarefas de forma a obter a melhor solução possível. Uma solução ótima é a que informa a sequência das tarefas a serem executadas por essa máquina que irá resultar no valor mínimo possível da soma dos atrasos de cada tarefa multiplicado pela respectiva penalidade. O problema do sequenciamento de tarefas em máquinas uniformes e paralelas (*Parallel Machines Total Weighted Tardiness Problem*), além de buscar o melhor sequenciamento dentro de cada máquina, busca a melhor distribuição possível de tarefas entre as diversas máquinas. Ou seja, o problema que envolve uma única máquina é uma parte do problema que envolve várias máquinas paralelas.

Cada tarefa pode ter outras características a serem consideradas no problema, que podem incluir, além do tempo de processamento, data de entrega e penalidade por atraso, a bonificação por entrega antes da data, tempo de preparação da máquina para cada tarefa, entre outras. As máquinas também podem ter características próprias ou serem todas iguais. Neste trabalho foi abordado o problema de sequenciamento de tarefas em máquinas paralelas e idênticas com tempo de preparação constante, ou seja, o *Parallel Machines Total Weighted Tardiness Problem*.

Para cada tarefa $i = 1,..., n$, é conhecido o seu tempo de processamento p_i, sua data de entrega d_i e o peso por dia de atraso da conclusão da tarefa em relação à sua data de entrega, w_i. O problema pode ser denotado por $Pm||\sum w_j T_j$ (DELLA CROCE; GARAIX; GROSSO, 2012). Cada problema considerado para os testes, conta com um conjunto independente de tarefas para serem particionadas entre as máquinas e sequenciada em cada uma delas.

1.2 Aplicações

Um exemplo de aplicação deste modelo é no sequenciamento de tarefas em processadores com vários núcleos, de forma a obter o menor tempo de processamento (BLAZEWICZ; DRABOWSKI; WEGLARZ, 1986). Também, pode-se citar como exemplo o planejamento da produção de uma empresa no nível operacional (ARENALES *et al.,* 2011), já que as aplicações industriais, em grande parte, exigem que o sequenciamento de tarefas seja realizado de forma que elas fiquem divididas entre máquinas e ordenadas dentro de cada uma delas, tal que torne a produção mais rápida e menos custosa. Outro exemplo é o agendamento da chegada de navios em um porto (KAO; LEE, 2007).

De forma geral, há vários tipos de problemas que ao serem abstraídos podem encontrar alguma relação com o problema de sequenciamento de tarefas em máquinas paralelas. Outros exemplos que podem ser citados são os ambientes de manufatura que podem contar com veículos autônomos, operadores de máquinas e outros fatores que afetem os processos (EDIS; OGUZ; OZKARAHAN, 2013) e a definição das tarefas que cada robô dentro de um grupo de robôs inteligentes deve realizar para chegar a um objetivo comum (CHENG; LI, 2010).

O problema aqui estudado é, segundo Brucker, Lenstra e Rinooy Kan (1977) e Du e Leung, (1990), do tipo *NP-Hard*. Isso significa que não se conhece algoritmo de tempo polinomial para encontrar a solução ótima do problema. Na prática, isto significa que, para problemas de grande porte, a utilização de métodos exatos se torna proibitiva do ponto de vista computacional, consumindo um tempo de processamento muito elevado. Desta forma, torna-se necessário o emprego de heurísticas e metaheurísticas de otimização para se obter soluções de boa qualidade em um tempo de processamento razoável.

1.3 Objetivos do Trabalho

Este trabalho tem como objetivo estudar e implementar a metaheurística *Iterated Local Search* ILS combinada com duas versões da técnica de busca local *Very Large-scale Neighborhood Search* – VLNS, para resolver o *Parallel Machines Total Weighted Tardiness Problem* para o qual existem relativamente poucos trabalhos desenvolvidos (DELLA CROCE; GARAIX; GROSSO, 2012).

Para o particionamento das tarefas entre as máquinas, foi testada uma busca VLNS ainda não testada para o problema, compondo a primeira versão da mateheurística ILS. A busca local desta versão utilizado um grafo de melhoria e um algoritmo de busca por ciclos negativos neste grafo para explorar a vizinhança (AHUJA *et al*., 2002). A segunda versão da busca VLNS, para o particionamento das tarefas, constrói um grafo específico para o problema e explora a vizinhança por meio de um algoritmo do *matching máximo* na rede (DELLA CROCE; GARAIX; GROSSO, 2012). Em ambas as versões, foi empregada a busca *Dynasearch*, que é uma VLNS para cada máquina. A *Dynasearch* utiliza um algoritmo de Programação Dinâmica para explorar a vizinhança e foi proposta por (CONGRAM; POTTS; VAN DE VELDE, 2002).

Os objetivos específicos foram: *i*) implementar a técnica de busca em vizinhança de grande porte *Dynasearch* para otimizar o sequenciamento das tarefas em cada máquina separadamente, e *ii*) implementar a busca em vizinhança de grande porte que utiliza o algoritmo de *matching máximo* para otimizar o sequenciamento das tarefas em um conjunto

de máquinas paralelas. O *Dynasearch* é uma variante do VLNS e foi proposta por (CONGRAM; POTTS; VAN DE VELDE, 2002) e cuja complexidade foi melhorada por Ergun e Orlin (2006). Nos testes computacionais, o *Dynasearch* obteve resultados superiores àqueles encontrados por técnicas clássicas de busca local (CONGRAM; POTTS; VAN DE VELDE, 2002).

1.4 Relevância do Trabalho

Este trabalho é relevante por apresentar originalidade ao estudar o *Parallel Machines Total Weighted Tardiness Problem* através de combinações das buscas *Dynasearch*, VLNS com *matching* e ciclo negativos, com a metaheurística ILS. Através deste estudo a exploração do desempenho desses métodos poderá ser melhor entendida.

1.5 Organização do Trabalho

Este trabalho está dividido da seguinte forma: na Seção 2 são apresentados o problema e os algoritmos relevantes para a resolução, na Seção 3 é apresentado o método de resolução do problema detalhando cada etapa de resolução. Na Seção 4 são apresentados os resultados dos testes computacionais e a Seção 5 contém as conclusões do trabalho.

2 REVISÃO BIBLIOGRÁFICA

Para problemas do tipo $Pm||\sum w_j T_j$ é dado um conjunto de tarefas $N = \{1, 2, ..., n\}$, assim como o tempo de processamento p_j, o peso w_j por atraso em relação à data de entrega, d_j especificada para cada tarefa $j \epsilon N$. As tarefas são processadas em uma sequência S em um conjunto $M = \{1, 2, ..., m\}$ de máquinas identificas e paralelas, de modo que suas completudes C_j, ou seja, a data em que cada tarefa j é finalizada, minimizem a função objetivo dada pela soma do atraso ponderado de cada tarefa, expresso em (1).

$$T(S) = \sum_{j=1}^{n} w_j T_j = \sum_{j=1}^{n} w_j max\{C_j - d_j, 0\} \quad (1)$$

Para $m = 1$ temos o problema de atraso total de uma única máquina, que é bem estudado e resolvido tanto com métodos exatos quanto métodos heurísticos. Para se ter uma visão geral de métodos de resolução de problemas de sequenciamento com uma única máquina, podem ser consultados os trabalhos de Bigras *et al*. (2008), Pan e Shi (2007), Rodrigues *et al*. (2008), Congram, Potts e van de Velde (2002) e Della Croce, Grosso e Tadei (2004). Os trabalhos mais recentes sobre máquinas paralelas e que também serviram de referências para o estudo foram os de Anghinolfi e Paolucci (2007), Bilge *et al*. (2004), Koulamas (1997) e Rodrigues *et al*. (2008).

Um dos problemas enfrentados na implementação de heurísticas de busca é como definir uma estrutura de vizinhança. Neste trabalho foi utilizada a técnica de busca em vizinhança de grande porte *Very Large-scale Neighborhood Search* (VLNS) com a detecção de ciclos negativos (AHUJA *et al.*, 2002) integrada com a metaheurística *Iterated Local Search* (ILS) (LOURENÇO; MARTIN; STÜTZLE, 2003) para a realização da busca entre as máquinas, como proposto por (DELLA CROCE; GARAIX; GROSSO, 2012). Esse procedimento foi testado anteriormente por Franco e Silva (2016), utilizando o método de busca local clássico *Best Improvement* para o sequenciamento das tarefas em cada máquina. Outro algoritmo estudado para a distribuição de tarefas entre máquinas é o proposto por Della Croce, Garaix e Grosso (2012), que também utiliza a técnica de busca em grande porte VLNS, mas nesta versão a verificação da vizinhança se dá por um algoritmo de *matching*, integrado ao ILS.

O particionamento das tarefas entre as máquinas foi realizado com o uso de duas técnicas. Na primeira implementação foi utilizado o VLNS desenvolvido por Franco e Silva (2016), que

constrói um *grafo de melhoria* e busca por ciclos negativos neste grafo. O VLNS é uma busca mais abrangente do que as buscas de realocação e troca de tarefas entre máquinas, que se baseiam em movimentos do tipo *2-optimal*. Esta técnica realiza os movimentos clássicos e ainda permite uma realocação ou troca de tarefas em cadeia, com as tarefas pertencentes a uma série de máquinas distintas, podendo inclusive envolver todas as máquinas do problema, sendo, portanto, do tipo *k-optimal*.

Na segunda implementação, foi desenvolvida a técnica de *matching* aliada ao VLNS para resolver a segunda parte do problema, dando uma base maior para testar e avaliar o desempenho de cada técnica. O *matching* é estudado na teoria dos grafos e também constrói um grafo de melhoria (BONDY; MURTY, 1976). Contudo, ao invés de procurar por ciclos negativos no grafo, o algoritmo realiza movimentos de tarefas entre as máquinas. Apenas duas máquinas são utilizadas por vez em cada movimento, sendo cada máquina chamada de vértice e o movimento realizado chamado de aresta (BONDY; MURTY, 1976). O algoritmo então salva a aresta e os vértices correspondentes, sempre que o movimento diminuir o atraso total do sistema, criando o grafo de melhoria. Então, ele seleciona arestas que não compartilham vértices entre si, mas que juntas ocasionem a maior diminuição do atraso total possível das máquinas. É importante ressaltar o fato de que as arestas selecionas não compartilham vértices, o que garante que o efeito do movimento de uma aresta do grafo não afeta o efeito de outra aresta. A consequência disso é que o efeito final dos movimentos pode ser calculado pela simples soma dos efeitos de cada aresta (BONDY; MURTY, 1976).

Neste trabalho, a busca VLNS também foi implementada para realizar a etapa de sequenciamento das tarefas em cada máquina individualmente. Isto foi obtido utilizando o algoritmo de Programação Dinâmica *Dynasearch*, proposto por Congram, Potts e van de Velde (2002) e melhorado por Ergun e Orlin (2006). Desta forma, tanto a etapa do particionamento das tarefas entre as máquinas, quanto o sequenciamento das tarefas em cada máquina são otimizados empregando busca de vizinhança de grande porte. Ambas as buscas são integradas na metaheurística ILS proposta por Della Croce, Garaix e Grosso (2012) para o problema de sequenciamento de tarefas em máquinas paralelas e uniformes.

Ao contrário dos algoritmos clássicos de busca, que realizam apenas uma troca por iteração, o *Dynasearch,* assim como os demais algoritmos de busca em vizinhança de grande porte (VLNS), podem realizar várias trocas por iteração, o que impede que a busca seja atraída para

ótimos locais pobres e pode encontrar soluções melhores do que os métodos tradicionais, quando executado separadamente em cada máquina (CONGRAM; POTTS; VAN DE VELDE, 2002). Além disso, em alguns problemas, o *Dynasearch* pode ser executado em tempo pseudo-polinomial (RODRIGUES *et al.,* 2008). Por isso, o *Dynasearch* foi combinado com a busca VLNS, para que ambas as etapas fossem otimizadas por uma técnica de busca em vizinhança de grande porte. O algoritmo foi testado com dados *benchmark* disponíveis na internet (http://algox.icomp.ufam.edu.br/index.php/benchmark-instances/weighted-tardiness-scheduling) e largamente utilizados com esta finalidade. Os resultados foram comparados com ótimo global, quando conhecido, e com aqueles obtidos por Franco e Silva (2016) para os problemas cujas soluções ótimas não são conhecidas.

3 DESENVOLVIMENTO

A metaheurística *Iterated Local Search* – ILS foi combinada com o *Dynasearch* e com as duas buscas VLNS, gerando as duas combinações estudadas:

1. ILS + *Dynasearch* + VLNS com ciclo negativo, que será denotado por (**ILS-DVLNS-cn**)

2. ILS + *Dynasearch* + VLNS com *matching*, doravante denotado por (**ILS-DVLNS-m**)

Foram realizados testes computacionais com problemas *benchmark* para avaliar a eficiência da implementação. Os resultados confirmaram a hipótese de que a utilização de buscas em vizinhança de grande porte nas duas etapas do problema pode aumentar a eficiência da metaheurística na resolução do problema.

Outra forma de se melhorar a eficiência da busca, é a aplicação de perturbações aleatórias. Isso permite que o algoritmo saia de ótimos locais e possibilita que ele encontre ótimos locais melhores ou até mesmo o ótimo global. Esta é uma característica da metaheurística ILS. Um problema com vários picos e vales é graficamente representado na Figura 1 - *Representação gráfica de problema com vários ótimos locais. Sem a realização de perturbações, os algoritmos ficam "presos" em um pico e não exploram o restante da função.*.

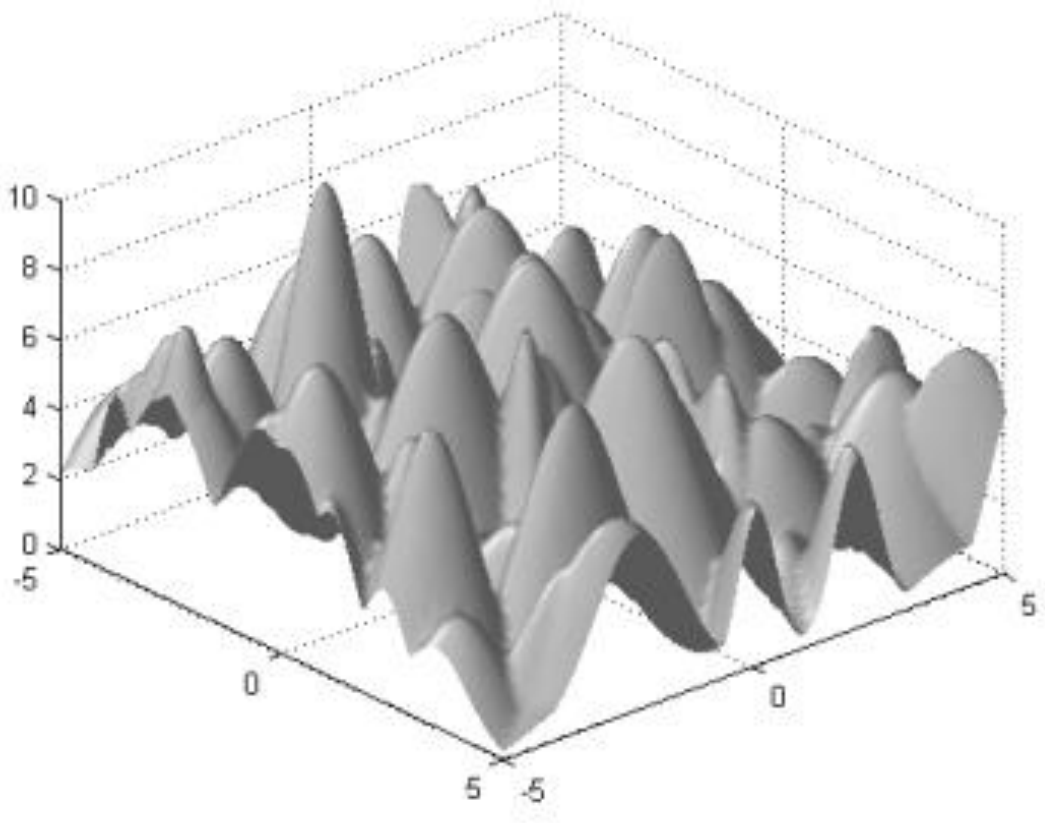

Figura 1 - Representação gráfica de problema com vários ótimos locais. Sem a realização de perturbações, os algoritmos ficam "presos" em um pico e não exploram o restante da função.
FONTE: GALLAGHER; YUAN, 2007

3.1 Particionamento das tarefas utilizando *VLNS* com ciclos negativos

Técnicas de busca local do tipo *2-optimal* são muito utilizadas para resolver problemas de otimização combinatória, entretanto, existem vizinhanças mais amplas que permitem realizar uma busca em um espaço muito maior de soluções. A técnica de busca VLNS é uma generalização da vizinhança de realocação e troca aos pares, onde um vizinho é obtido através da realização de uma *troca cíclica* ou de uma *troca em cadeia*. A vizinhança de troca cíclica envolve a troca aos pares, mas também permite que tarefas de três ou mais máquinas estejam envolvidas na troca. Considerando o caso de três máquinas e de troca cíclica, uma tarefa da máquina 1 é realocada para a máquina 2, que por sua vez tem uma tarefa realocada para a máquina 3, que finalmente tem uma tarefa realocada para a máquina 1. No caso da troca em cadeia, a máquina 3 não realoca qualquer tarefa para a máquina 1 e após o movimento a

máquina 1 terá uma tarefa a menos, assim como a máquina 3 contará com uma tarefa além do que havia anteriormente.

Para que seja possível realizar uma busca na vizinhança de troca cíclica e em cadeia, de forma eficiente, é necessário construir um grafo direcionado que represente a vizinhança de uma dada solução $\boldsymbol{S}$. Tal grafo, denominado *grafo de melhoria* $G(\boldsymbol{S})$, é definido para cada solução factível $\boldsymbol{S}$ do problema. Considerando n tarefas e m máquinas, seja $S[j]$ a máquina que contém a tarefa j, então, $G(\boldsymbol{S}) = (\boldsymbol{N}, \boldsymbol{E})$ é uma grafo direcionado com o conjunto de nós $\boldsymbol{N}$ contendo um nós para cada tarefa $i = 1, .., n$, um nó para cada máquina $j = n+1, ..., n+m$, e um super-nó $t = n+m+1$. O conjunto $\boldsymbol{E}$ contém os seguintes tipos de arcos:

1. (i, j) de uma tarefa i para outra j que representa a substituição da tarefa j pela tarefa i na máquina $S[j]$ que contém a tarefa j. O custo deste tipo de arco é dado por $c(\{i\} \cup S[j] \setminus \{j\}) - c(S[j])$;

2. (i, m_j) de uma tarefa i para uma máquina m_j, sendo $m_j \neq S[i]$, que representa a inserção da tarefa i na máquina m_j sem que qualquer tarefa seja retirada da máquina m_j. Neste caso, o custo do arco é calculado pela expressão $c(\{i\} \cup m_j) - c(m_j)$;

3. (t, j) do super-nó t para cada tarefa j, que considera a retirada da tarefa j da máquina em que se encontra sem que qualquer outra tarefa seja inserida nesta máquina. O custo deste arco é dado por $c(S[j] \setminus \{j\}) - c(S[j])$;

4. (m_j, t) de cada máquina m_j para o super-nó t, para permitir a formação de ciclos. Esse tipo de arco tem custo zero.

Um ciclo negativo, ou seja, um ciclo com custo negativo em $G(\boldsymbol{S})$ corresponde a uma troca cíclica ou em caminho que melhora o valor da função objetivo referente à solução corrente. Esta é uma maneira eficiente de realizar uma busca por soluções pertencentes à vizinhança de S que melhoram a solução corrente. Portanto, basta encontrar um ciclo negativo no grafo direcionado $G(\boldsymbol{S})$ para se obter um vizinho melhor. Neste trabalho foi implementado o algoritmo de *Walk to the root* (Cherkassky e Goldberg 1999) que tem complexidade da ordem de $O(n^2m)$, sendo n o número de tarefas e m o número de máquinas, para encontrar um ciclo negativo ou detectar que o grafo não contém qualquer ciclo negativo.

Uma vez encontrado um ciclo negativo, as trocas são realizadas, a solução é atualizada assim como o seu respectivo grafo de melhoria. O processo é repetido até que nenhum ciclo negativo seja encontrado no grafo de melhoria. Quando o grafo não apresentar qualquer ciclo negativo, então a solução corrente é uma solução ótima local segundo esta vizinhança. Esta heurística de busca local foi implementada e testada por Franco e Silva (2016) que traz mais detalhes para o leitor interessado.

No exemplo da Figura 2, considerando detectado o ciclo negativo destacado, a sua "atualização" corresponde à troca da tarefa 9 pela 14 na máquina acima, da tarefa 2 pela tarefa 9 na máquina à direita e acima, da tarefa 3 pela tarefa 2 na máquina à direita e abaixo e da tarefa 14 pela tarefa 3 na máquina à esquerda. Observe que uma máquina não foi envolvida na troca, mas ela também poderia estar envolvida dependendo da configuração do ciclo negativo detectado.

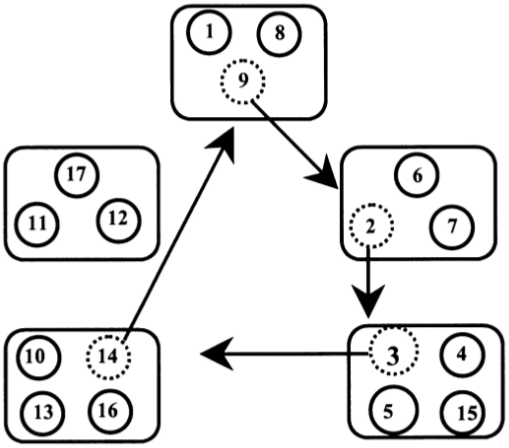

Figura 2 - Exemplo de ciclo negativo envolvendo 4 das 5 máquinas de um determinado problema.

FONTE: AHUJA *et al.*, 2002

3.2 Particionamento utilizando VLNS e *matching*

A teoria dos grafos é aqui utilizada com a aplicação do algoritmo de *matching* para realizar uma busca do tipo VLNS. Para construir o grafo de melhoria, são feitos movimentos que realocam ou trocam uma dada tarefa de uma máquina para uma outra. Segundo Della Croce, Garaix e Grosso (2012), para uma dada solução $\boldsymbol{S}$, seleciona-se um par de máquinas, sendo que estas terão, respectivamente, as sequências de tarefas σ_1 e σ_2, permitindo assim os seguintes movimentos:

(a) extrair uma tarefa j de σ_1 e inseri-la em σ_2

(b) extrair uma tarefa j de σ_2 e inseri-la em σ_1

(c) extrair tarefas $i \in \sigma_1$, $j \in \sigma_2$ e inserir i em $\sigma_2 - \{j\}$ e j em $\sigma_1 - \{i\}$

Após explorar todos os movimentos possíveis utilizando (a), (b) e (c) nas sequências σ_1 e σ_2 é possível definir a vizinhança, cujo tamanho é não-polinomial em relação ao número de máquinas, assim como pesquisar esta vizinhança em tempo polinomial seguindo o algoritmo proposto por Della Croce, Garaix e Grosso (2012) dado por:

(1) para cada par de máquinas (m_1, m_2), calcular a diminuição máxima do atraso Δ_{m_1,m_2} que é obtida ao se realizar os movimentos (a), (b) e (c) nas sequências σ_1, σ_2, para todo $j \in \{\sigma_1\} \cup \{\sigma_2\}$.

(2) construir o grafo de melhoria $G(M,E)$ onde:

$$E = \{\{m_1, m_2\} : \Delta_{m_1,m_2} > 0\}$$

(3) a próxima solução vizinha será obtida ao aplicar o *matching* com peso máximo em G e executar os movimentos relacionados às arestas selecionadas.

É importante ressaltar que a realização dos movimentos (a), (b) e (c) necessita de $O(n^3)$ operações e que, de forma geral, a exploração dessa vizinhança é extremamente rápida e barata em termos computacionais, sendo essa técnica bastante útil principalmente quando o número de máquinas aumenta (DELLA CROCE; GARAIX; GROSSO, 2012).

O problema de *matching* com peso máximo consiste em selecionar as arestas de um grafo, tal que a soma dos pesos das arestas selecionadas seja máxima e que quaisquer duas arestas não tenham nós em comum (BONDY; MURTY, 1976). As arestas selecionadas determinam um emparelhamento entre os seus nós extremos. O emparelhamento é considerado perfeito quando todos os vértices do grafo são emparelhados (BONDY; MURTY, 1976). Um número par de vértices é necessário para isto. Quando o grafo possui um número ímpar de vértices, não ocorrerá emparelhamento perfeito. Entretanto, nesse último caso, quando as arestas são emparelhadas de forma que não é possível selecionar alguma outra aresta para se emparelhar, o emparelhamento é considerado máximo (BONDY; MURTY, 1976).

Como o algoritmo do *matching* não seleciona arestas que podem piorar o custo da solução corrente, é garantido que sempre que o problema testado por ele possuir um número par de máquinas, poderá ocorrer o emparelhamento perfeito (BONDY; MURTY, 1976). Também é garantido que para qualquer número de máquinas, em todos os problemas testados com qualquer número de tarefas, ocorrerá, quando não houver emparelhamento perfeito, o emparelhamento máximo (BONDY; MURTY, 1976). Isso garante que o algoritmo consiga realizar a máxima melhoria possível em todo grafo gerado pelos passos (1), (2) e (3) através dos movimentos (a), (b) e (c), descritos no início desta sessão. Realizar a máxima melhoria possível é de grande importância, pois isso faz com que o problema convirja para um ótimo local mais rápido, diminuindo os custos computacionais e o tempo de processamento.

No exemplo da Figura 3 tem-se um grafo formado por uma rede com 5 máquinas. Cada máquina é representada por um vértice e as arestas representam um dos movimentos (a), (b) ou (c) realizado entre as máquinas. Nesse exemplo, a aresta vermelha e a azul foram selecionadas pelo algoritmo do *matching* para realizar os respectivos movimentos. Já a Figura 4 mostra com mais detalhes a interação entre as tarefas de duas máquinas, com o algoritmo de *matching* escolhendo como aresta a que representa uma troca de duas tarefas entre as máquinas.

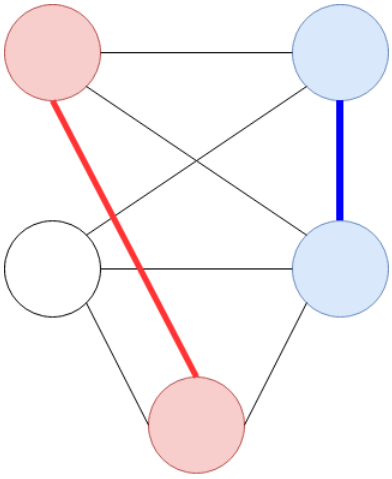

Figura 3 - Exemplo de grafo formado por uma rede com o algoritmo do matching selecionando as máquinas para realizar os movimentos (Elaboração própria)

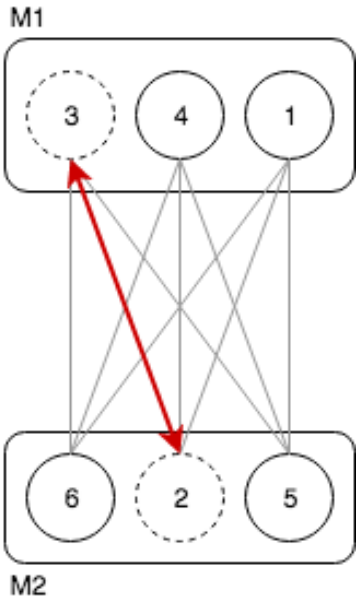

Figura 4 - Grafo de melhoria com duas máquinas. A melhor aresta é a indicada pela seta vermelha, que realiza o movimento de troca de duas tarefas entre as máquinas (Elaboração própria)

O algoritmo desenvolvido neste trabalho implementou os passos (a), (b) e (c) e os movimentos (1), (2), (3) como descritos por Della Croce, Garaix e Grosso (2012), sendo Bondy e Murty (1976) utilizado para consulta e melhor entendimento de como desenvolver a solução com matching máximo.

3.3 Vizinhança *Dynasearch* em uma máquina

Para o problema de sequenciamento de tarefas em uma máquina, o *Swap* é uma vizinhança *2-optimal* que troca a posição de duas tarefas quaisquer independentemente de elas serem adjacentes ou não. Verificar a otimalidade local para uma vizinhança do tipo *k-optimal* requer $\Omega(n^k)$ operações, onde n corresponde ao número de tarefas do problema. Para valores baixos de k, a vizinhança *k-optimal* pode ser pesquisada rapidamente utilizando os métodos clássicos de busca, mas à medida que k aumenta, a busca se torna impraticável. Desta forma, utiliza-se a busca *2-optimal*, que correspondem à vizinhança *Swap* para problemas de sequenciamento.

Seja $\sigma = (\sigma(1), ..., \sigma(n))$ uma permutação ou sequência que define a ordem corrente de processamento das tarefas de uma máquina, sendo $\sigma(i)$ a tarefa na posição i, para $i = 1,..., n$. A vizinhança *Swap* de uma permutação σ compreende todas as sequências que podem ser obtidas pela troca de quaisquer duas tarefas $\sigma(i)$ e $\sigma(j)$, onde $1 \leq i < j \leq n$. A vizinhança do tipo *Dynasearch* de σ permite uma nova permutação obtida por uma série de *trocas independentes*. Dois movimentos que trocam a tarefa $\sigma(i)$ com $\sigma(j)$ e a tarefa $\sigma(k)$ com $\sigma(l)$, são ditos independentes se $max\{i , j\} < min\{k , l\}$ ou $min\{i , j\} > max\{k , l\}$. A vizinhança *Dynasearch* consiste de todas as soluções que podem ser obtidas a partir de σ por uma série de pares de movimentos de trocas independentes. Esta vizinhança tem tamanho $2^{n-1} - 1$, ou seja, de tamanho exponencial.

Uma forma de encontrar o melhor conjunto de trocas independentes de uma permutação σ pode ser obtido, de maneira eficiente, utilizando um algoritmo de Programação Dinâmica. Este algoritmo adota o esquema de enumeração *forward* no qual as tarefas são adicionadas ao final da sequência parcial corrente e são possivelmente trocadas de posição. Define-se que para uma sequência parcial estar no estado (k, σ), $k = 1,..., n$, ela pode ser obtida da sequência parcial $(\sigma(1), ..., \sigma(k))$ aplicando uma série de movimentos independentes. Para encontrar a melhor sequência na vizinhança *Dynasearch* de σ, que define o estado (n, σ), é necessário encontrar uma sequência que tem valor objetivo mínimo entre todas as sequências neste estado. Este é o princípio do algoritmo de Programação Dinâmica utilizado para encontrar o melhor vizinho da estrutura *Dynasearch*.

Seja σ_k a sequência parcial com a soma mínima dos atrasos ponderados para as tarefas $\sigma(1)$, ..., $\sigma(k)$ entre todas as sequências parciais no estado (k, σ). Além disto, seja $F(\sigma_k)$ a soma dos

atrasos ponderados para as tarefas $\sigma(1)$, ..., $\sigma(k)$ em σ_k. Esta sequência parcial é obtida a partir de uma outra sequência parcial σ_i que tem valor objetivo mínimo entre todas sequências parciais em algum estado prévio (i, σ), onde $0 \leq i < k$, adicionando a tarefa $\sigma(k)$ no final da sequência se $i = k - 1$, ou adicionando primeiro as tarefas $\sigma(i + 1)$, ..., $\sigma(k)$ e então trocando de posição as tarefas $\sigma(i + 1)$ e $\sigma(k)$ se $0 \leq i < k - 1$. Esta recursividade é utilizada na implementação do algoritmo de Programação Dinâmica até que seja obtida a melhor solução de uma vizinhança *Dynasearch*. O processo é repetido para uma série de soluções iniciais distintas, guardando sempre a melhor solução encontrada.

3.4 O Algoritmo de Programação Dinâmica

Considere uma dada solução inicial $\sigma = (\sigma(1), ..., \sigma(n))$ a partir da qual é realizada uma busca local do tipo *Dynasearch*. A nova solução será construída a partir de σ realizando uma série de movimentos independentes até que nenhuma melhoria seja alcançada. O Algoritmo de Programação Dinâmica a ser aplicado recursivamente é descrito a seguir.

Seja $F(\sigma_k)$ o atraso total ponderado das tarefas $\sigma(1)$, ..., $\sigma(k)$ em σ_k. Assim, temos a seguinte inicialização:

$$F(\sigma_0) = 0$$

$$F(\sigma_1) = w_{\sigma(1)}\left(p_{\sigma(1)} - d_{\sigma(1)}\right)^+$$

Onde $(x)^+ = max\{0, x\}$. Então, a recursão para $k = 2, ..., n$ é dada por:

$$F(\sigma_k) = min\left\{\begin{array}{l} F(\sigma_{k-1}) + w_{\sigma(k)}\left(P_{\sigma(k)} - d_{\sigma(k)}\right)^+, \\ \displaystyle\min_{0 \leq i \leq k-2}\left\{\begin{array}{c} F(\sigma_i) + w_{\sigma(k)}\left(P_{\sigma(i)} + p_{\sigma(k)} - d_{\sigma(k)}\right)^+ \\ + \displaystyle\sum_{j=i+2}^{k-1} w_{\sigma(j)}\left(P_{\sigma(j)} + p_{\sigma(k)} - p_{\sigma(i+1)} - d_{\sigma(j)}\right)^+ \\ + w_{\sigma(i+1)}\left(P_{\sigma(k)} - d_{\sigma(i+1)}\right)^+ \end{array}\right\} \end{array}\right\}$$

Então, o valor da solução ótima é igual a $F(\sigma_n)$, e a sequência correspondente pode ser obtida por um processo do tipo *backtracking*. Por exemplo, se o valor de $F(\sigma_k)$ for dado pelo primeiro termo na minimização, então a tarefa $\sigma(n)$ não é trocada de posição em relação à solução corrente, e prosseguimos para saber como o valor de $F(\sigma_{n-1})$ foi determinado. Caso contrário, se o valor de $F(\sigma_k)$ for dado pelo segundo termo, para algum índice *i*, então as tarefa das posições $\sigma(n)$ e $\sigma(i+1)$ devem ser trocadas na nova solução, e prosseguimos para saber como o valor de $F(\sigma_i)$ foi determinado. O processo se repete até que $F(\sigma_0)$ ou $F(\sigma_1)$ apareça no lado direito da equação de recursão, quando todos os *swaps* independentes são identificados e o procedimento de *backtracking* termina.

A implementação do algoritmo de programação dinâmica com busca na vizinhança *Dynasearch* inicia com uma solução inicial σ^0 obtida por algum método guloso. Durante a iteração (t, σ^{t-1}) é a solução corrente, que se pretende melhorar realizando um movimento na vizinhança *Dynasearch*. Usando o algoritmo de Programação Dinâmica apresentado, é possível calcular o valor de $F(\sigma_k^{t-1})$ para $k = 1, ..., n$, e então aplica-se um procedimento do tipo *backtracking* para encontrar a correspondente sequência σ^t.

A solução definida por σ^t é um ótimo local na vizinhança *Dynasearch* se $F(\sigma_n^{t-1}) = F(\sigma_n^{t-2})$, sendo que $F(\sigma_n^{-1})$ representa o valor objetivo da solução inicial σ^0. Neste caso, o algoritmo é encerrado. Por outro lado, se $F(\sigma_n^{t-1}) \leq F(\sigma_n^{t-2})$ então deve ser executada uma nova iteração tendo σ^t como a solução corrente.

O algoritmo não encerra a sua execução quando $F(\sigma_n^{t-1}) = F(\sigma_n^{t-2})$. A forma de continuar com a execução utilizada é a realização de uma perturbação aleatória em $F(\sigma_n^{-1})$. Essa perturbação muda $F(\sigma_n^{-1})$, piorando a solução. Entretanto, a partir desta perturbação o algoritmo de Programação Dinâmica pode encontrar um novo ótimo local melhor do que $F(\sigma_n^{-1})$. Esse mecanismo de perturbação aleatória possibilita então que em várias execuções e em execuções por longos períodos de tempo, soluções diferentes sejam encontradas a partir da mesma solução inicial.

3.5 Geração da Solução Inicial

Para solucionar o problema estudado, é necessário, para os algoritmos desenvolvidos, que o problema já se encontre em um estado no qual toda tarefa esteja alocada em uma máquina. A partir deste estado inicial, chamado de solução inicial, pode-se aplicar os algoritmos estudados, que procuram melhorar a soluções inicial/solução corrente do problema.

A solução inicial, utilizada neste trabalho, foi obtida empregando a estratégia do *Early Due Date* – EDD (BAKER, 1984), uma heurística gulosa que consiste em ordenar as tarefas priorizando aquelas com menor data de entrega. Após a ordenação monotonicamente crescente pela data de entrega das tarefas, deve ser verificada no final de qual máquina a tarefa de menor data de entrega deve ser alocada, de tal forma que incorra no menor custo possível. A tarefa é alocada no final da máquina que estiver disponível mais cedo e o processo se repete até que todas as tarefas tenham sido alocadas.

3.6 Desenvolvimento e Validação da Heurística *Dynasearch*

Após a geração da solução inicial, o algoritmo de programação dinâmica que utiliza a vizinhança *Dynasearch* é aplicado individualmente para cada máquina. A validação dele se deu a partir de problemas simples que podiam ser resolvidos com o auxílio de planilhas e também se deu com a comparação com os resultados obtidos por Congram, Potts e van de Velde (2002).

Os testes mostraram que a heurística produziu os resultados esperados na resolução do exemplo apresentado por Congram, Potts e van de Velde (2002). O exemplo consistia de uma máquina com as tarefas i, os respectivos tempos de processamento p_i, peso w_i e data de entrega d_i apresentados na Tabela 1.

Tabela 1 - Tarefas a serem alocadas em uma única máquina (Adaptado de Congram, Potts e van de Velde (2002)).

Tarefa	1	2	3	4	5	6
Tempo de processamento p	3	1	1	5	1	5
Peso w	3	5	1	1	4	4
Data de entrega d	1	5	3	1	3	1

A heurística foi iniciada com a solução inicial $\sigma^0 = \{1,2,3,4,5,6\}$, com atraso total ponderado igual a 109. A partir desta solução, foi executado o algoritmo de Programação Dinâmica com o objetivo de minimizar o atraso total ponderado. Após 3 iterações, o algoritmo terminou sua execução, obtendo atraso total com peso igual a 67. As trocas realizadas a cada iteração podem ser vistas na Tabela 2.

Tabela 2 - Exemplo de execução do algoritmo de Programação Dinâmica em uma única máquina

Iteração	*Sequência*	*Atraso Total com Peso*
	1 2 3 4 5 6	109
1	1 3 2 5 4 6	89
2	1 5 2 3 6 4	68
3	5 1 2 3 6 4	67

Esse mesmo problema ao ser resolvido utilizando a heurística clássica *Best Improvement* chegou ao atraso total de 70 após 3 iterações, ou seja, obteve um resultado pior do que o *Dynasearch* (CONGRAM; POTTS; VAN DE VELDE, 2002) como mostra a Tabela 3. Esse comportamento deve se repetir para outros problemas com o *Dynasearch* se saindo melhor por fazer movimentações melhores e convergir mais rápido para um ótimo.

Tabela 3 - Exemplo de execução do algoritmo *Best Improvement* em uma única máquina

Iteração	*Sequência*	*Atraso Total com Peso*
	1 2 3 4 5 6	109
1	1 2 3 5 4 6	90
2	1 2 3 5 6 4	75
3	5 2 3 1 6 4	70

Outros testes foram realizados com esse algoritmo e verificou-se que ele funcionava como descrito por Congram, Potts e van de Velde (2002), obtendo melhores resultados do que algoritmos de busca que realizam uma única troca por iteração.

3.7 Desenvolvimento do ILS-DVLNS-cn para o Problema $Pm||\sum w_j T_j$

Com o *Dynasearch* implementado e validado, a etapa que se segue é a implementação do VLNS com ciclo negativo. A partir da solução inicial fornecida pelo *Earliest Due Date*, o algoritmo desenvolvido gera um grafo de melhoria com os arcos criados conforme os procedimentos descritos no item 3.1 **Particionamento das tarefas utilizando *VLNS* com ciclos negativos**.

Então, o algoritmo encontra um ciclo negativo que proporciona uma diminuição do atraso total da solução corrente e realiza os movimentos relativos a esse ciclo, chegando a uma nova solução. Com isso, o sistema é transformado e se aproxima do seu ótimo local. O processo se repete até que o grafo de melhoria não tenha qualquer ciclo negativo. Esta última solução é uma solução ótima local na vizinhaça de grande porte.

Quando a busca VLNS atinge uma solução ótima local, o ILS provoca perturbações aleatórias na solução corrente, gerando uma nova solução a partir da qual é realizada uma nova busca local do tipo VLNS.

3.8 Desenvolvimento do ILS-DVLNS-m para o Problema $Pm||\sum w_j T_j$

Outra etapa que se faz necessária após a implementação do *Dynasearch*, é a implementação da segunda versão do ILS, que utiliza a busca VLNS com *matching.* A solução inicial para este algoritmo também parte do resultado do *Earliest Due Date.* É importante lembrar que o VLNS com ciclo negativo e o VLNS com *matching* são duas técnicas diferentes de busca em vizinhança de grande porte, que têm os resultados estudados e comparados neste trabalho.

O desenvolvimento do algoritmo seguiu a lógica já descrita no item 3.2. O programa desenvolvido realiza os movimentos indicados por Della Croce, Garaix e Grosso (2012) na solução corrente e gera a rede de melhoria aplicando neste o algoritmo de *matching* máximo. A partir daí, consegue selecionar os movimentos que devem ser aplicados na solução corrente para melhora-la.

Assim como o VLNS com ciclo negativo, o processo se repete até que não seja possível encontrar um *matching* que melhore a solução corrente, o que impede com que o algoritmo continue a explorar o espaço de soluções por se encontrar em um ponto de ótimo local. Então, o algoritmo ILS usado com o VLNS com *matching* também realiza perturbações aleatórias nessas soluções, com o objetivo de criar soluções que possam convergir para outros ótimos locais presentes no espaço de soluções do problema.

3.9 Desenvolvimento de funções para realizar perturbações aleatórias nas soluções

A realização de perturbações aleatórias em soluções que são ótimos locais é uma característica do ILS e permite que uma maior parte do espaço de soluções seja explorado.

Isso possibilita a obtenção de soluções melhores do que aqueles encontrados anteriormente. Neste trabalho, foram utilizados dois tipos de perturbações.

O primeiro tipo de perturbação foi do tipo intra-máquinas. O algoritmo desenvolvido para este fim realiza modificações nas posições das tarefas em máquinas escolhidas aleatoriamente. Isso gera uma nova solução que possibilita a existência de ciclos negativos e arestas que, ao serem exploradas, podem melhorar a melhor solução encontrada. Além disso, a nova ordem das tarefas pode fazer com que o *Dynasearch* também convirja para uma melhor solução.

O segundo tipo de perturbação utilizado foi do tipo inter-máquinas. Essa perturbação consiste em aplicar movimentos de troca de tarefas escolhidas entre pares de máquinas tomadas aleatoriamente. Essa perturbação também permite que tanto o *Dynasearch,* quanto VLNS com ciclo negativo e o VLNS com *matching* ajudem a encontrar uma solução melhor do que a melhor solução conhecida até o momento.

3.10 Desenvolvimento da Metaheurística ILS para o VLNS com ciclo negativo

A metaheurística ILS é inicializada com a solução gerada pelo método *EDD*() na linha 1, e conta com as duas heurísticas de busca local: a busca em N_1, realizada pela função *dynasearch*(), na linha 4 do Algoritmo 1, tenta melhorar o sequenciamento das tarefas de cada máquina isoladamente pela vizinhança *Dynasearch*. A busca em N_2, feita pela função *vlns_ciclo_negativo*() na linha 6 do Algoritmo 1, faz a otimização considerando movimentos entre máquinas feita pela busca do tipo *n-optimal*, ou seja, utilizando a busca VLNS com ciclo negativo considerando movimentos entre máquinas. É nesta função que os ciclos negativos são detectados de forma a melhorar a solução corrente. Nas linhas 19 e 22 ocorre a perturbação intra-máquinas aplicando a função *kick1*( ). Posteriormente, na linha 22, é realizada a perturbação inter-máquinas na função *kick2*( ).

Iterated Local Search (N_1, N_2, T(), EDD(), Tempo, MaxSemMelhora)

1. S* = S = EDD();

```
2.    IterCont = 0;
3.    enquanto tempo de processamento < Tempo faça
4.       S1 = dynasearch(S, N1);
5.       Repita
6.          S2 = vlns_ciclo_negativo(S1, N2);
7.          se(T(S2) < T(S1) então
8.             S1 = S2;
9.          fim_se;
10.      até que N2 não melhore S1
11.      se T(S1) < T(S*) então
12.         S* = S1;
13.         interCont = 0;
14.      senão
15.         interCont = interCont + 1;
16.      fim_se;
17.      se N2 falhar em melhorar S1 então
18.         se iterCont > MaxSemMelhora então
19.            S = kick1(S*);
20.            interCont = 0;
21.         senão
22.            S = kick2(S2);
23.         fim_se;
24.      fim_se;
25.   fim_enquanto;
retorna S*;
```

Figura 5 - Pseudocódigo da implementação do ILS com o VLNS e ciclo negativo.

3.11 Desenvolvimento da Metaheurística ILS para o VLNS com matching

A integração do VLNS com *matching* ao ILS e ao *Dynasearch* foi feito de forma semelhante ao que ocorreu com o VLNS com ciclo negativo. Como as condições de parada, as funções de perturbação, a geração da solução inicial e o algoritmo de programação dinâmica são o mesmo para ambos, a única diferença se dá na linha 6 do algoritmo, onde a função *vlns_ciclo_negativo()* é substituída pela função *vlns_matching().*

```
Iterated Local Search (N1, N2, T(), EDD(), Tempo, MaxSemMelhora)

1.    S* = S = EDD();
2.    IterCont = 0;
3.    enquanto tempo de processamento < Tempo faça
```

```
4.      S1 = dynasearch(S, N1);
5.      Repita
6.          S2 = vlns_matching(S1, N2);
7.          se(T(S2) < T(S1) então
8.              S1 = S2;
9.          fim_se;
10.     até que N2 não melhore S1
11.     se T(S1) < T(S*) então
12.         S* = S1;
13.         interCont = 0;
14.     senão
15.         interCont = interCont + 1;
16.     fim_se;
17.     se N2 falhar em melhorar S1 então
18.         se iterCont > MaxSemMelhora então
19.             S = kick1(S*);
20.             interCont = 0;
21.         senão
22.             S = kick2(S2);
23.         fim_se;
24.     fim_se;
25.  fim_enquanto;
retorna S*;
```

Figura 6 - Pseudocódigo da implementação do ILS com VLNS e *matching*.

3.12 Realização de testes de validação das metaheurísticas com ciclo negativo e com *matching*

Os algoritmos desenvolvidos na implementação das metaheurísticas utilizando *matching* e ciclo negativo foram validados por meio de testes utilizando problemas de pequeno porte, para os quais são conhecidas as soluções ótimas. Com isso, *bugs* foram detectados e corrigidos e o código foi otimizado.

3.13 Realização de testes de validação e desempenho

Os algoritmos de busca entre máquinas (VLNS com ciclo negativo e VLNS com *matching*) foram combinados com o *Dynasearch*. A última etapa de validação foi então realizada com a execução dos algoritmos utilizando problemas *benchmark* disponíveis na internet com soluções ótimas conhecidas. Além disso, a partir dessa etapa começaram os testes de eficiência dos algoritmos, sendo que eles foram testados com os mais variados problemas com ótimos conhecidos. Os algoritmos aqui estudados tiveram os resultados comparados entre si e com os resultados obtidos no trabalho de Franco e Silva (2016) que é uma implementação do ILS com o particionamento das tarefas entre as máquinas realizado pela busca VLNS e o sequenciamento das tarefas em cada máquina é feito utilizando a heurística clássica de *Best Improvement*, ao invés do *Dynasearch* explorado neste trabalho.

Também foram realizados testes em problemas sem solução ótima conhecida. Nessa etapa, também há comparações com os resultados do ILS-VLNS de Franco e Silva (2016) e entre os algoritmos aqui estudados.

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Todos os algoritmos descritos foram implementados na linguagem de programação C/C++. O foco das implementações recaiu na eficiência computacional dos algoritmos, a fim de conseguir resultados de qualidade em pouco tempo e com um baixo consumo de recursos computacionais.

Para gerar a solução inicial, foi utilizada a heurística gulosa *Earliest Due Date*, considerando as datas de entrega de todas as máquinas para a alocação de tarefas. O *hardware* utilizado para resolver os problemas foi um computador com processador *Intel (R) Pentium (R) N3540 @* 2.16 GHz e 4 GB de memória RAM. Inicialmente foram realizados testes com problemas cuja solução ótima é conhecida, verificando desta forma a eficiência das metaheurísticas propostas. Posteriormente foram utilizados problemas maiores cujas soluções ótimas não são conhecidas. Os resultados foram comparados, sempre que possível, com aqueles obtidos por todas as versões implementadas do ILS.

Nota-se que as perturbações do ILS pioram a solução encontrada para que, a partir desta nova solução, as técnicas de busca tenham a possibilidade de encontrar alguma solução ainda melhor. Como resultado, após realizar as perturbações aleatórias, os algoritmos podem chegar a resultados diferentes a partir da mesma solução inicial cada vez que são executados. Nem sempre isso acontece, devido à natureza aleatória das perturbações, por isso nos testes realizados neste trabalho cada programa foi executado 5 vezes, sendo que cada execução teve duração de 15 minutos. Esse número de execuções, aliado ao tempo elevado de execução permitiu que uma grande área do espaço de soluções de cada problema fosse explorada.

4.1 Resultados dos testes da Metaheurística ILS - Dynasearch + VLNS com ciclo negativo

A *Dynasearch* foi incorporada ao algoritmo ILS para a realização de testes, sendo que o VLNS foi executado na etapa de busca entre máquinas e o *Dynasearch* executado na etapa de

busca dentro de cada máquina, compondo assim a Metaheurística ILS + *Dynasearch* + VLNS com ciclo negativo. Utilizando a Metaheurística ILS + *Dynasearch* + VLNS com ciclo negativo*,* denominada doravante apenas ILS-DVLNS-cn, primeiro foram resolvidos 10 problemas, cada um contando com 50 tarefas e 4 máquinas. Posteriormente foram resolvidos outros 10 problemas, contendo 50 tarefas e 10 máquinas, para os quais não são conhecidas as soluções ótimas. Os resultados foram comparados com aqueles obtidos pela versãp ILS-VLNS implementada por Franco e Silva (2016). As instâncias desta primeira etapa de testes foram obtidas da adaptação de um conjunto de problemas com uma única máquina disponíveis na *OR-library* (BEASLEY, 1990).

Conforme pode ser observado na Tabela 4, para 4 máquinas o ILS-DVLNS-cn encontrou a melhor média para 7 problemas, contra 2 problemas cujas melhores médias foram encontradas pelo ILS-VLNS, além de 1 empate.

A estrutura de tabela apresentada na Tabela 4 é aqui explicada. Cada problema foi executado 5 vezes com cada Metaheurística. As colunas Mínimo mostram o menor atraso ponderado obtido por cada método, a colunas Máximo mostram o maior atraso ponderado obtido por cada método, as colunas Média mostram as médias aritméticas dos atrasos obtidos nas 5 execuções de cada problema. Os melhores resultados médios são destacados em negrito. Já as colunas DP mostram o desvio padrão resultante dessas 5 execuções.

Tabela 4 - Resultados obtidos utilizando ILS-VLNS e ILS-DVLNS-cn para problemas com n = 50 e m = 4, sem soluções ótimas conhecidas

	ILS-DVLNS-cn				ILS-VLNS			
Problema	Mínimo	Máximo	Média	DP	Mínimo	Máximo	Média	DP
10	3434,0	3434,0	**3434,0**	0,00	3434	3435	3434,4	0,55
20	20992,0	20999,0	20995,2	2,77	20986	20990	**20987,6**	1,82
30	59,0	59,0	**59,0**	0,00	59	61	60,0	1,00
40	8423,0	8433,0	**8427,4**	3,85	8428	8443	8434,8	5,89
50	55755,0	55761,0	**55758,4**	2,30	55757	55763	55759,2	2,28
60	2287,0	2289,0	**2287,4**	0,89	2288	2293	2289,6	1,95
70	22984,0	23027,0	**23000,0**	17,22	22982	23014	23002,8	12,52
80	0,0	0,0	0,0	0,00	0	0	0,0	0,00
90	6315,0	6340,0	6331,4	10,01	6324	6333	**6329,2**	3,70

100	34433,0	34462,0	**34446,2**	10,55	34440	34461	34451,2	9,58

Para problemas com 10 máquinas, cujos resultados são apresentados na Tabela 5, o ILS-DVLNS-cn encontrou a melhor média para 5 problemas, enquanto o ILS-VLNS encontrou a melhor média para 3 problemas, além de 2 empates. Para esses testes o ILS-DVLNS-cn mostrou ser mais eficiente para problemas com 4 máquinas, ou seja, aqueles que contam com maior número de tarefas por máquina. Por outro lado, para problemas com um número maior de máquinas, o ILS-DVLNS-cn teve uma ligeira redução na sua eficiência.

Tabela 5 - Resultados obtidos utilizando ILS-VLNS e ILS-DVLNS-cn para problemas com n = 50 e m = 10, sem soluções ótimas conhecidas

	ILS-DVLNS-cn				ILS-VLNS			
Problema	Mínimo	Máximo	Média	DP	Mínimo	Máximo	Média	DP
10	2113	2121	**2118,4**	3,21	2115	2123	2119,2	3,49
20	11369	11371	11369,8	0,84	11364	11373	**11369,0**	3,39
30	165	167	**165,8**	0,84	165	168	166,8	1,30
40	5635	5653	5641,4	7,54	5630	5643	**5637,4**	5,60
50	27568	27570	27568,8	1,10	27568	27570	27568,8	1,10
60	1908	1931	**1917,8**	10,08	1918	1961	1940,8	17,24
70	13162	13171	**13166,4**	3,51	13164	13173	13169,0	3,54
80	0	0	0,0	0,00	0	0	0,0	0,00
90	5386	5401	5392,2	6,30	5381	5402	**5389,4**	8,53
100	17807	17812	**17810,2**	2,17	17806	17812	17810,4	2,51

No segundo grupo de testes, inicialmente o ILS-DVLNS-cn foi executado com um conjunto de problemas obtidos em http://algox.icomp.ufam.edu.br/index.php/benchmark-instances/weighted-tardiness-scheduling, para resolver 25 problemas, cujas soluções ótimas são conhecidas. Todos esses problemas contam com 20 tarefas e 4 máquinas. Os resultados obtidos foram comparados com aqueles obtidos pelo ILS-VLNS. Ambos os métodos de busca obtiveram resultados idênticos, apresentados na Tabela 6. Essa semelhança de resultados dificultou a identificação dos tipos de problema para os quais cada versão da metaheurística

ILS tem seu melhor desempenho. Para buscar um padrão de comportamento do ILS-DVLNS-cn, ele foi executado para outros problemas cujas soluções ótimas são conhecidas.

Nas tabelas 6, 9, 10, 11, 12, 13 e 14 as colunas R e T que aparecem nas tabelas se referem aos parâmetros de distribuição uniforme utilizada na geração de cada problema. As colunas %otm se referem à diferença percentual entre a solução ótima do problema e a melhor solução encontrada pelo respectivo algoritmo de busca utilizado. DP mostra o desvio padrão das 5 soluções encontradas nas execuções do algoritmo para cada problema. A coluna QT indica o número de vezes que solução ótima foi encontrada pelo algoritmo.

Tabela 6 - Resultados obtidos utilizando ILS-VLNS e ILS-DVLNS-cn para problemas com n = 20 e m = 4, com soluções ótimas conhecidas

Problemas			ILS – VLNS				ILS-DVLNS-cn			
#	R	T	%otm	Média	DP	QT	%otm	Média	DP	QT
1	0,2	0,2	0,00	144	0	5	0,00	144	0	5
2	0,2	0,4	0,00	48	0	5	0,00	48	0	5
3	0,2	0,6	0,00	0	0	5	0,00	0	0	5
4	0,2	0,8	0,00	0	0	5	0,00	0	0	5
5	0,2	1,0	0,00	0	0	5	0,00	0	0	5
6	0,4	0,2	0,00	487	0	5	0,00	487	0	5
7	0,4	0,4	0,00	386	0	5	0,00	386	0	5
8	0,4	0,6	0,00	301	0	5	0,00	301	0	5
9	0,4	0,8	0,00	300	0	5	0,00	300	0	5
10	0,4	1,0	0,00	324	0	5	0,00	324	0	5
11	0,6	0,2	0,00	1069	0	5	0,00	1069	0	5
12	0,6	0,4	0,00	1033	0	5	0,00	1033	0	5
13	0,6	0,6	0,00	1036	0	5	0,00	1036	0	5
10	0,6	0,8	0,00	1070	0	5	0,00	1070	0	5
15	0,6	1,0	0,00	887	0	5	0,00	887	0	5
16	0,8	0,2	0,00	1866	0	5	0,00	1866	0	5
17	0,8	0,4	0,00	1904	0	5	0,00	1904	0	5
18	0,8	0,6	0,00	1681	0	5	0,00	1681	0	5
19	0,8	0,8	0,00	1461	0	5	0,00	1461	0	5
20	0,8	1,0	0,00	1261	0	5	0,00	1261	0	5
21	1,0	0,2	**0,53**	2655	0	0	**0,53**	2655	0	0
22	1,0	0,4	**0,46**	2392	0	0	**0,46**	2392	0	0
23	1,0	0,6	0,00	2150	0	5	0,00	2150	0	5
24	1,0	0,8	0,00	1904	0	5	0,00	1904	0	5
25	1,0	1,0	0,00	1681	0	5	0,00	1681	0	5

4.2 Resultados dos testes da Metaheurística ILS + *Dynasearch* + *VLNS matching*

A heurística *Dynasearch* também foi incorporada ao algoritmo ILS, integrado ao VLNS com *matching*, ou seja, Metaheurística ILS + *Dynasearch* + VLNS com *matching*. Utilizando a metaheurística ILS + *Dynasearch* + VLNS com *matching,* denominada doravante apenas ILS-DVLNS-m, foram resolvidos os mesmos problemas resolvidos pelos ILS-DVLNS-cn. Para os problemas sem ótimo conhecido, os resultados foram comparados com o ILS-VLNS implementado por Franco e Silva (2016). As instâncias desta primeira etapa de testes também foram obtidas da adaptação de um conjunto de problemas com uma única máquina disponíveis na página http://people.brunel.ac.uk/~mastjjb/jeb/info.html. Esta página, denominada *OR-library*, reúne e disponibiliza os dados de uma série de problemas de otimização, os quais são largamente utilizados para testar algoritmos propostos para os respectivos problemas.

Conforme pode ser observado na Tabela 7, para 4 máquinas o ILS-DVLNS-m encontrou a melhor média para 6 problemas, contra 2 problemas cujas melhores médias foram encontradas pelo ILS-VLNS, além de 2 empates. Os melhores resultados médios são destacados em negrito.

Tabela 7 - Resultados obtidos utilizando ILS-VLNS e ILS-DVLNS-m para problemas com n = 50 e m = 4, sem soluções ótimas conhecidas

	ILS-DVLNS-m				ILS-VLNS			
Problema	Mínimo	Máximo	Média	DP	Mínimo	Máximo	Média	DP
10	3434	3434	**3434.0**	0.00	3434	3435	3434.4	0.55
20	20985	20993	20988.8	3.37	20986	20990	**20987.6**	1.82
30	59	59	**59.0**	0.00	59	61	60.0	1.00
40	8420	8430	**8425.6**	3.26	8428	8443	8434.8	5.89
50	55757	55760	**55758.4**	1.20	55757	55763	55759.2	2.28
60	2287	2287	**2287.0**	0.00	2288	2293	2289.6	1.95
70	22977	23024	23002.8	16.29	22982	23014	23002.8	12.52
80	0	0	0.0	0.00	0	0	0.0	0.00
90	6322	6351	6336.2	10.42	6324	6333	**6329.2**	3.70
100	34432	34445	**34438.8**	4.71	34440	34461	34451.2	9.58

Para problemas com 10 máquinas, cujos resultados são apresentados na Tabela 8, o ILS-DVLNS-m encontrou a melhor média para 8 problemas, enquanto o ILS-VLNS não encontrou a melhor média em nenhum deles, além de 2 empates. Para esses testes o ILS-DVLNS-m mostrou ser mais eficiente do que o ILS-VLNS para problemas com 10 máquinas, ao contrário do ILS-DVLNS-cn que apresentou uma diminuição na eficiência. Ou seja, aqueles problemas que contam com maior número de máquinas são melhor explorados pelo ILS-DVLNS-m.

Tabela 8 - Resultados obtidos utilizando ILS-VLNS e ILS-DVLNS-m para problemas com n = 50 e m = 10, sem soluções ótimas conhecidas

	ILS-DVLNS-m				ILS-VLNS			
Problema	Mínimo	Máximo	Média	DP	Mínimo	Máximo	Média	DP
10	2112	2119	**2115**	2.28	2115	2123	2119.2	3.49
20	11361	11367	**11362.8**	2.23	11364	11373	11369	3.39
30	165	165	**165**	0.00	165	168	166.8	1.30
40	5624	5629	**5626**	1.90	5630	5643	5637.4	5.60
50	27568	27572	27568.8	1.60	27568	27570	27568.8	1.10
60	1876	1918	**1902**	15.56	1918	1961	1940.8	17.24
70	13162	13171	**13166.6**	3.83	13164	13173	13169	3.54
80	0	0	0	0.00	0	0	0	0.00
90	5378	5395	**5388.8**	6.37	5381	5402	5389.4	8.53
100	17805	17811	**17808**	2.28	17806	17812	17810.4	2.51

No segundo grupo de testes, inicialmente o ILS-DVLNS-m também foi executado com um conjunto de problemas obtidos em http://algox.icomp.ufam.edu.br/index.php/benchmark-instances/weighted-tardiness-scheduling, no mesmo molde do ILS-DVLNS-cn. Os resultados obtidos para os problemas de 50 tarefas e 4 máquinas foram comparados com aqueles obtidos pelo ILS-VLNS. Os resultados estão apresentados na Tabela 9. Os testes foram realizados em 15 problemas. O ILS-DVLNS-m conseguiu alcançar o ótimo em todas as instâncias testadas, enquanto o ILS-VLNS não encontrou o ótimo em um problema.

Tabela 9 - Resultados obtidos utilizando ILS-VLNS e ILS-DVLNS-m para problemas com n = 20 e m = 4, com soluções ótimas conhecidas

Problemas			ILS – VLNS				ILS-DVLNS-m			
#	R	T	%otm	Média	DP	QT	%otm	Média	DP	QT
1	0,2	0,2	0,00	144	0	5	0,00	144	0	5

2	0,2	0,6	0,00	0	0	5	0,00	0	0	5
3	0,2	1,0	0,00	0	0	5	0,00	0	0	5
4	0,4	0,2	0,00	487	0	5	0,00	487	0	5
5	0,4	0,6	0,00	301	0	5	0,00	301	0	5
6	0,4	1,0	0,00	324	0	5	0,00	324	0	5
7	0,6	0,2	0,00	1069	0	5	0,00	1069	0	5
8	0,6	0,6	0,00	1036	0	5	0,00	1036	0	5
9	0,6	1,0	0,00	887	0	5	0,00	887	0	5
10	0,8	0,2	0,00	1866	0	5	0,00	1866	0	5
11	0,8	0,6	0,00	1681	0	5	0,00	1681	0	5
12	0,8	1,0	0,00	1261	0	5	0,00	1261	0	5
13	1,0	0,2	**0,53**	2655	0	0	0,00	2641	0	5
14	1,0	0,6	0,00	2150	0	5	0,00	2150	0	5
15	1,0	1,0	0,00	1681	0	5	0,00	1681	0	5

4.3 Comparação entre o ILS-DVLNS-cn e o ILS-DVLNS-m

Todos os problemas dos testes seguintes contam com 20 tarefas, sendo que os problemas da Tabela 10 contam com 2 máquinas, os problemas da Tabela 11 contam 4 máquinas, os problemas da Tabela 12 contam com 6 máquinas, os da Tabela 13 contam com 8 máquinas e finalmente, na Tabela 14 são apresentados os resultados dos problemas que utilizam 10 máquinas para sequenciar as 20 tarefas. Para todos os problemas, o ILS-DVLNS-cn foi executado 5 vezes e os resultados obtidos foram comparados com os ótimos e são apresentados a seguir. Estas instâncias não foram resolvidas pela versão ILS-VLNS, portanto, não foi possível realizar o mesmo tipo de comparação.

O ILS-DVLNS-cn conseguiu chegar à solução ótima na maioria dos problemas com 2 e 4 máquinas (Tabela 10 e Tabela 11) e em todos os problemas com 6 máquinas (Tabela 12). Contudo, alcançou o ótimo em apenas 6 dos 15 problemas com 8 máquinas (Tabela 13) e em apenas 8 dos 15 problemas com 10 máquinas (Tabela 14). Este resultado pode ser um indício de que uma quantidade muito baixa de tarefas por máquinas pode comprometer o desempenho das ILS-DVLNS-cn.

Já o ILS-DVLNS-m conseguiu alcançar o ótimo em todos os problemas testados em todas as 5 execuções de cada problema. Como as duas metaheurísticas utilizam o *Dynasearch*, percebe-se que a queda de desempenho do ILS-DVLNS-cn com o aumento do número de máquinas e a diminuição do número de tarefas por máquina se deve à execução da busca entre máquinas utilizando o ciclo negativo.

Tabela 10 - Resultados obtidos utilizando ILS- DVLNS-cn e ILS-DVLNS-m para problemas com n = 20 e m = 2, com soluções ótimas conhecidas

Problemas			ILS-DVLNS-cn				ILS-DVLNS-m			
#	R	T	%otm	Média	DP	QT	%otm	Média	DP	QT
1	0,2	0,2	0	147	0	5	0	147	0	5
2	0,2	0,6	0	0	0	5	0	0	0	5
3	0,2	1,0	0	0	0	5	0	0	0	5
4	0,4	0,2	0	695	0	5	0	695	0	5
5	0,4	0,6	0	341	0	5	0	341	0	5
6	0,4	1,0	0	227	0	5	0	227	0	5
7	0,6	0,2	0	1741	0	5	0	1741	0	5
8	0,6	0,6	**0.45**	1557	0	0	0	1550	0	5
9	0,6	1,0	0	1290	0	5	0	1290	0	5
10	0,8	0,2	0	3186	0	5	0	3186	0	5
11	0,8	0,6	0	2768	0	5	0	2768	0	5
12	0,8	1,0	**0.15**	1997	0	0	0	1994	0	5
13	1,0	0,2	**0.04**	4653	0	0	0	4651	0	5
14	1,0	0,6	0	3667	0	5	0	3667	0	5
15	1,0	1,0	0	2768	0	5	0	2768	0	5

Tabela 11 - Resultados obtidos utilizando ILS- DVLNS-cn e ILS-DVLNS-m para problemas com n = 20 e m = 4, com soluções ótimas conhecidas

Problemas			ILS-DVLNS-cn				ILS-DVLNS-m			
#	R	T	%otm	Média	DP	QT	%otm	Média	DP	QT
1	0,2	0,2	0,00	144	0	5	0,00	144	0	5
2	0,2	0,6	0,00	0	0	5	0,00	0	0	5
3	0,2	1,0	0,00	0	0	5	0,00	0	0	5
4	0,4	0,2	0,00	487	0	5	0,00	487	0	5
5	0,4	0,6	0,00	301	0.89	5	0,00	301	0	5
6	0,4	1,0	0,00	324	0	5	0,00	324	0	5

7	0,6	0,2	0,00	1069	0	5	0,00	1069	0	5
8	0,6	0,6	0,00	1036	0	5	0,00	1036	0	5
9	0,6	1,0	0,00	887	0	5	0,00	887	0	5
10	0,8	0,2	0,00	1866	0	5	0,00	1866	0	5
11	0,8	0,6	0,00	1681	0	5	0,00	1681	0	5
12	0,8	1,0	0,00	1261	0	5	0,00	1261	0	5
13	1,0	0,2	**0,53**	2655	0	0	0,00	2641	0	0
14	1,0	0,6	0,00	2150	0	5	0,00	2150	0	5
15	1,0	1,0	0,00	1681	0	5	0,00	1681	0	5

Tabela 12 - Resultados obtidos utilizando ILS- DVLNS-cn e ILS-DVLNS-m para problemas com n = 20 e m = 6, com soluções ótimas conhecidas

Problemas			ILS-DVLNS-cn				ILS-DVLNS-m			
#	R	T	%otm	Média	DP	QT	%otm	Média	DP	QT
1	0,2	0,2	0	161	0	5	0	161	0	5
2	0,2	0,6	0	16	0	5	0	16	0	5
3	0,2	1,0	0	28	0	5	0	28	0	5
4	0,4	0,2	0	435	0	5	0	435	0	5
5	0,4	0,6	0	358	0	5	0	358	0	5
6	0,4	1,0	0	388	0	5	0	388	0	5
7	0,6	0,2	0	884	0	5	0	884	0	5
8	0,6	0,6	0	888	0	5	0	888	0	5
9	0,6	1,0	0	766	0	5	0	766	0	5
10	0,8	0,2	0	1458	0	5	0	1458	0	5
11	0,8	0,6	0	1325	0	5	0	1325	0	5
12	0,8	1,0	0	1048	0	5	0	1048	0	5
13	1,0	0,2	0	1994	0	5	0	1994	0	5
14	1,0	0,6	0	1643	0	5	0	1643	0	5
15	1,0	1,0	0	1325	0	5	0	1325	0	5

Tabela 13 - Resultados obtidos utilizando ILS- DVLNS-cn e ILS-DVLNS-m para problemas com n = 20 e m = 8, com soluções ótimas conhecidas

Problemas			ILS-DVLNS-cn				ILS-DVLNS-m			
#	R	T	%otm	Média	DP	QT	%otm	Média	DP	QT
1	0,2	0,2	0	177	0	5	0	177	0	5

2	0,2	0,6	0	77	0	5	0	77	0	5
3	0,2	1,0	0	100	0	5	0	100	0	5
4	0,4	0,2	0	439	0	5	0	439	0	5
5	0,4	0,6	**0.05**	422.2	0.45	4	0	422	0	5
6	0,4	1,0	0	431	0	5	0	431	0	5
7	0,6	0,2	**2.2**	790	0	0	0	773	0	5
8	0,6	0,6	**0.12**	823	0	0	0	822	0	5
9	0,6	1,0	**0.27**	734	0	0	0	732	0	5
10	0,8	0,2	**0.55**	1282	0	0	0	1275	0	5
11	0,8	0,6	**0.43**	1168	0	0	0	1163	0	5
12	0,8	1,0	**0.105**	953	0	0	0	952	0	5
13	1,0	0,2	0	1678	0	5	0	1678	0	5
14	1,0	0,6	**0.5**	1420	0	0	0	1413	0	5
15	1,0	1,0	**0.43**	1168	0	0	0	1163	0	5

Tabela 14 - Resultados obtidos utilizando ILS- DVLNS-cn e ILS-DVLNS-m para problemas com n = 20 e m = 10, com soluções ótimas conhecidas

Problemas			ILS-DVLNS-cn				ILS-DVLNS-m			
#	R	T	%otm	Média	DP	QT	%otm	Média	DP	QT
1	0,2	0,2	**2.56**	200	0	0	0.00	195	0	5
2	0,2	0,6	0.00	161	0	5	0.00	161	0	5
3	0,2	1,0	0.00	177	0	5	0.00	177	0	5
4	0,4	0,2	0.00	476	0	5	0.00	476	0	5
5	0,4	0,6	0.00	458	0	5	0.00	458	0	5
6	0,4	1,0	0.00	482	0	5	0.00	482	0	5
7	0,6	0,2	**0.26**	765	0	0	0.00	763	0	5
8	0,6	0,6	0.00	799	0	5	0.00	799	0	5
9	0,6	1,0	0.00	732	0	5	0.00	732	0	5
10	0,8	0,2	**0.69**	1161	0	0	0.00	1153	0	5
11	0,8	0,6	**1.30**	1093	0	0	0.00	1079	0	5
12	0,8	1,0	**0.11**	898	0	0	0.00	897	0	5
13	1,0	0,2	0.00	1483	0	5	0.00	1483	0	5
14	1,0	0,6	**0.63**	1280	0	0	0.00	1272	0	5
15	1,0	1,0	**1.30**	1093	0	0	0.00	1079	0	5

A partir dos dados das Tabelas 10 a 14, foi montado o gráfico da Figura 7 que tem no eixo das abscissas o número de máquinas e nas ordenadas o número de vezes que o ILS-DVLNS-cn e o ILS-DVLNS-m encontram a solução ótima. Pode-se observar, através dos dados, que o

número de soluções ótimas encontradas pelo ILS-DVLNS-cn aumentou quando o número de máquinas aumento de 2 para 4 e depois de 4 para 6. Posteriormente, houve uma queda acentuada quando o número de máquinas passou de 6 para 8 e uma leve recuperação quando passou para 10. Observando a *linha de tendência* que foi gerada utilizando os cinco pontos do gráfico tem-se uma reta com inclinação negativa, indicando que o aumento no número de máquinas pode acarretar em uma diminuição no número de soluções ótimas encontradas. O ILS-DVLNS-m encontrou o ótimo de todos os problemas.

Desta forma, há evidências que o ILS-DVLNS-cn obtém seus melhores resultados nos problemas em que cada máquina deve realizar um grande número de tarefas. Resta destacar que, embora o ILS-DVLNS-cn não tenha encontrado a solução ótima para a maioria dos problemas com 20 tarefas e 8 máquinas, as soluções obtidas são muito próximas dos valores ótimos, variando entre 0,05% e 2,2% do ótimo e em média 0,52% da solução ótima. Já o ILS-DVLNS-m aparenta ser eficaz em todo tipo de cenário, independentemente da quantidade de tarefas em cada máquina.

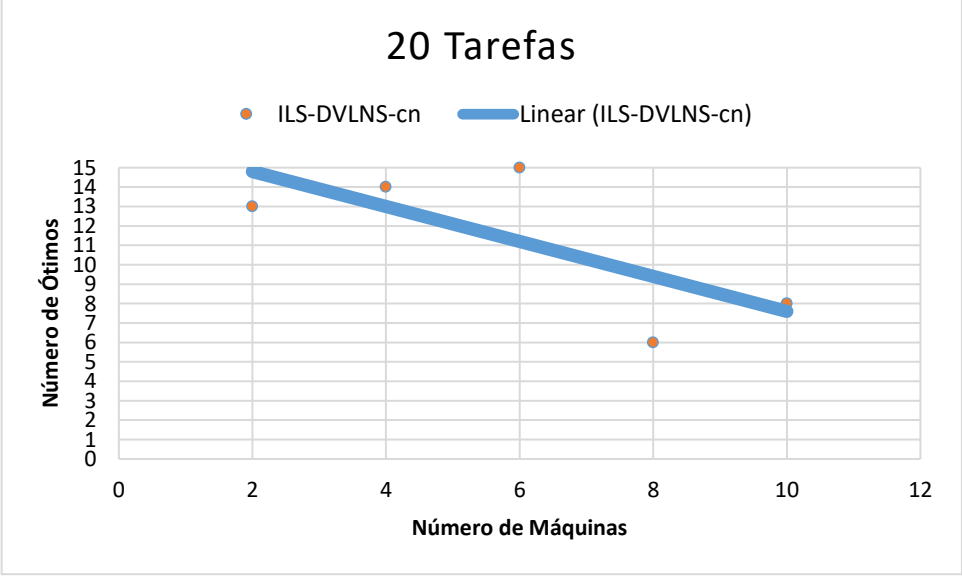

Figura 7 - Gráfico com linha de tendência do número de ótimos encontrados pelo ILS-DVLNS-cn para 20 tarefas em 2, 4, 6, 8 e 10 máquinas

Também foi possível comparar os resultados obtidos nos problemas sem ótimos conhecidos, com 50 tarefas para 4 e 10 máquinas para o ILS-DVLNS-cn e o ILS-DVLNS-m. Dos 10 problemas testados com 4 máquinas o ILS-DVLNS-m foi melhor em 4, perdendo em 2 e empatando em 4, como pode ser visto na Tabela 15. Já para os problemas com 10 máquinas, o ILS-DVLNS-cn foi melhor em 7 deles, empatando em 2 e perdendo em 1 para o ILS-

DVLNS-cn, como consta na Tabela 16. Este é outro indício de que a diminuição do número de tarefas por máquina prejudica a eficiência da versão ILS-DVLNS-cn.

Tabela 15 - Resultados obtidos utilizando ILS- DVLNS-cn e ILS-DVLNS-m para problemas com n = 50 e m = 4, sem soluções ótimas conhecidas

	ILS-DVLNS-cn				ILS-DVLNS-m			
Problema	Mínimo	Máximo	Média	DP	Mínimo	Máximo	Média	DP
10	3434	3434	3434	0.00	3434	3434	3434.0	0.00
20	20992	20999	20995	2.77	20985	20993	**20989**	3.37
30	59	59	59	0.00	59	59	59.0	0.00
40	8423	8433	8427	3.85	8420	8430	**8426**	3.26
50	55755	55761	55758	2.30	55757	55760	55758.4	1.20
60	2287	2289	2287	0.89	2287	2287	**2287**	0.00
70	22984	23027	**23000**	17.22	22977	23024	23002.8	16.29
80	0	0	0	0.00	0	0	0.0	0.00
90	6315	6340	**6331**	10.01	6322	6351	6336.2	10.42
100	34433	34462	34446.2	10.55	34432	34445	**3443**	4.71

Tabela 16 - Resultados obtidos utilizando ILS-DVLNS-cn e ILS-DVLNS-m para problemas com n = 50 e m = 10

	ILS-DVLNS-cn				ILS-DVLNS-m			
Problema	Mínimo	Máximo	Média	DP	Mínimo	Máximo	Média	DP
10	2113	2121	2118	3.21	2112	2119	**2115**	2.28
20	11369	11371	11370	0.84	11361	11367	**11363**	2.23
30	165	167	166	0.84	165	165	**165**	0.00
40	5635	5653	5641	7.54	5624	5629	**5626**	1.90
50	27568	27570	27569	1.10	27568	27572	27568.8	1.60
60	1908	1931	1918	10.08	1876	1918	**1902**	15.56
70	13162	13171	**13166**	3.51	13162	13171	13166.6	3.83
80	0	0	0	0.00	0	0	0	0.00
90	5386	5401	5392.2	6.30	5378	5395	**53889**	6.37
100	17807	17812	17810.2	2.17	17805	17811	**17808**	2.28

Com os resultados obtidos nos testes indicados nas Tabelas 9 a 16, montou-se o gráfico da Figura 8, que compara o ILS-DVLNS-cn com o ILS-DVLNS-m relacionando a porcentagem de problemas em cada metaheurística se saiu melhor do que a outra, no eixo das ordenadas, com a razão de tarefas por máquina, no eixo das abscissas.

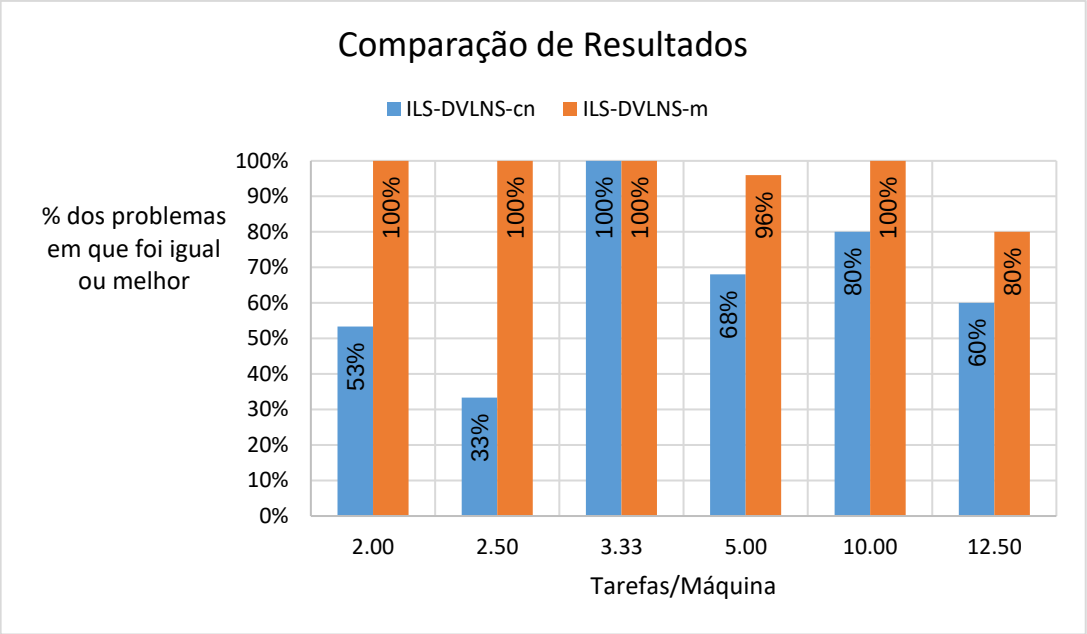

Figura 8 – Gráfico com a relação entre razão de Tarefas/Máquina relacionado à porcentagem de vezes que cada metaheurística alcançou empate ou foi melhor do que a outra

Também é possível analisar o valor mínimo (melhor solução) encontrado por cada metaheurística em cada problema sem ótimos conhecidos. Para os problemas com 50 tarefas e 4 máquinas o ILS-DVLNS-m foi o melhor em 4, o ILS-DVLNS-cn foi o melhor em 2, havendo empate nas outras 4 instâncias como pode ser observado na Tabela 17.

Tabela 17 - Relação dos mínimos encontrados pelos três métodos para problemas com n = 50 e m = 4

	ILS-DVLNS-cn		ILS-VLNS		ILS-DVLNS-m	
Problema	Mínimo	DP	Mínimo	DP	Mínimo	DP
10	3434	0,00	3434	0,55	3434	0,00
20	20992	2,77	20986	1,82	**20985**	3,37
30	59	0,00	59	1,00	59	0,00
40	8423	3,85	8428	5,89	**8420**	3,26
50	**55755**	2,30	55757	2,28	55757	1,20
60	2287	0,89	2288	1,95	2287	0,00
70	22984	17,22	22982	12,52	**22977**	16,29
80	0	0,00	0	0,00	0	0,00
90	**6315**	10,01	6324	3,70	6322	10,42
100	34433	10,55	34440	9,58	**34432**	4,71

Para os problemas com 50 tarefas e 10 máquinas o ILS-DVLNS-m foi o melhor em 6 ocorrendo empate nas outras 4 instâncias como pode ser observado na Tabela 18.

Tabela 18 - Relação dos mínimos encontrados pelos três métodos para problemas com n = 50 e m = 10

	ILS-DVLNS-cn		ILS-VLNS		ILS-DVLNS-m	
Problema	Mínimo	DP	Mínimo	DP	Mínimo	DP
10	2113	3,21	2115	3,49	**2112**	2,28
20	11369	0,84	11364	3,39	**11361**	2,23
30	165	0,84	165	1,30	165	0,00
40	5635	7,54	5630	5,60	**5624**	1,90
50	27568	1,10	27568	1,10	27568	1,60
60	1908	10,08	1918	17,24	**1876**	15,56
70	13162	3,51	13164	3,54	13162	3,83
80	0	0,00	0	0,00	0	0,00
90	5386	6,30	5381	8,53	**5378**	6,37
100	17807	2,17	17806	2,51	**17805**	2,28

5. PRODUTOS

Este trabalho gerou como produtos publicações em congressos e submissão para publicação em revista. Ele foi publicado no XLIX Simpósio Brasileiro de Pesquisa Operacional, 2017 com o título *O uso de técnicas de busca em vizinhança de grande porte para o Problema de Sequenciamento de Tarefas em Máquinas Paralelas* (FERREIRA; SILVA, 2017). Também foi publicado em 2018 no XIX Latin-Iberoamerican Conference on Operation Research - CLAIO 2018 com o título *Uma Busca em Vizinhança de Grande Porte para o Sequenciamento de Tarefas em Máquinas Paralelas* (FERREIRA, SILVA, 2018). Além disso, o trabalho foi enviado para publicação na revista *Gestão da Produção Operação e Sistemas* (GEPROS), onde já foi revisado.

6. CONCLUSÕES

Neste trabalho foram empregadas heurísticas de busca em vizinhança de grande porte nas duas etapas do problema. O VLNS com: *i*) ciclo negativo e *ii*) com o *matching* foram utilizados para realizar a alocação das tarefas às máquinas. O *Dynasearch* foi usado para definir a melhor sequência das tarefas em cada máquina, em ambos os casos. A combinação do ILS com a busca *Dynasearch* e o VLNS com o ciclo negativo resultou no chamado ILS-DVLNS-cn enquanto a combinação do ILS com a busca *Dynasearch* e o VLNS com o *matching* gerou o chamado ILS-DVLNS-m. Essas versões foram comparadas entre si e com a versão ILS-VLNS, que usa busca em vizinhança de grande porte com ciclo negativo apenas na primeira etapa, empregando a busca clássica do tipo *Best Improvement* na segunda etapa.

Utilizando a metaheurística ILS-DVLNS-cn para resolver problemas com 50 tarefas, cujas soluções não são conhecidas, foi verificado que este teve um desempenho acentuadamente melhor do que o ILS-VLNS com 4 máquinas, sendo que para 10 máquinas a sua eficiência sofreu uma ligeira redução. O melhor desempenho no caso de 4 máquinas se dá pelo motivo de que, quando há um número maior de tarefas por máquina, o *Dynasearch* consegue realizar o sequenciamento com muito mais eficácia do que a heurística *Best Improvement* utilizada no ILS-VLNS. Por outro lado, ao analisar problemas com 20 tarefas e 4 máquinas, com soluções ótimas conhecidas, não se obteve diferença significativa entre as duas metaheurísticas, com as duas versões atingindo a solução ótima na maioria dos problemas. Apesar do objetivo das metaheurísticas ser o de encontrar a solução ótima, nesse caso o comportamento eficaz de ambas as versões dificulta a identificação de condições que faz com que uma versão seja mais eficiente do que a outra.

Ao investigar a eficiência do ILS-DVLNS-cn em relação ao número de tarefas por máquinas, é possível perceber que essa versão da ILS foi bastante eficiente para resolver problemas com 20 tarefas e com 2, 4 e 6 máquinas, sendo que para os problemas com 6 máquinas o ILS-DVLNS-cn conseguiu encontrar a solução ótima em todas as execuções. Entretanto, ao aumentar o número de máquinas para 8 e para 10, a qualidade da solução caiu consideravelmente, com a solução ótima sendo alcançada em apenas 6 dos 15 problemas testados. Isso reforça a tese de que aumentar o número de máquina diminui a eficiência do ILS-DVLNS-cn. Não obstante, o método foi capaz de obter soluções muito próximas do valor

ótimo, com um *gap* médio de 0,52% para este caso. Assim, pode-se concluir que o ILS-DVLNS-cn é bastante competitivo, produzindo resultados muito próximos das soluções ótimas.

Também não é esperado que o ILS-VLNS consiga desempenho melhor do que o ILS-DVLNS-cn em problemas com menor número de tarefas por máquina, já que a estratégia de busca entre as máquinas é a mesma e a estratégia de sequenciamento *Dynasearch* em cada máquina do ILS-DVLNS-cn é mais eficaz. Isso pode ser observado pela diferença de desempenho dos dois algoritmos ao aumentar o número de tarefas por máquina. A diferença dos resultados se dá principalmente pela diferença entre os algoritmos de sequenciamento *Dyanasearch* e *Best Improvement*, já que ambos utilizam a mesma estratégia de distribuição de tarefas entre máquinas. Na validação do *Dynasearch* também foi observada sua superioridade em relação à heurística de busca clássica *Best Improvement*. Então, se conclui que, devido ao *Dynasearch*, sempre que o número de tarefas por máquina for elevado, o ILS-DVLNS-cn conseguirá bons resultados. O único problema observado no ILS-DVLNS-cn é que o uso de memória pelo algoritmo é muito grande quando o número de tarefas e máquinas aumenta. Isso acontece porque o tamanho da rede gerada para se encontrar o ciclo negativo cresce bastante tanto com o número de tarefas quanto com o número de máquinas.

Sobre o ILS-DVLNS-m, a análise dos resultados permite concluir que, apesar de utilizar uma técnica de emparelhamento simples e computacionalmente barata, o algoritmo de *matching* faz com que o ILS-DVLNS-m alcance melhores resultados do que a técnica de detecção de ciclos negativos, utilizada no ILS-DVLNS-cn. Como ambas as metaheurísticas utilizam o mesmo algoritmo de sequenciamento dentro de cada máquina, o *Dynasearch*, conclui-se que a diferença de desempenho se dá nos algoritmos utilizados para busca entre as máquinas. Ou seja, para os problemas testados, o algoritmo de *matching* se mostrou mais eficaz. Por montar grafos pequenos, o algoritmo consome menos memória, o que permite que problemas maiores sejam executados sem limitações de memória computacional.

Pode-se concluir também que o Dynasearch foi o método mais eficaz para buscas em uma única máquina para os problemas testados, se comparado ao método clássico de Best Improvement, pois consegue encontrar ótimos de melhor qualidade. Também se conclui que a busca de grande porte entre máquinas paralelas e uniformes é mais eficiente quando é utilizando o *matching*, do que utilizando a detecção de ciclos negativos. Observou-se que para

os problemas testados o *matching* apresentou resultados melhores. Estes resultados mostram que a versão ILS-DVLNS-m conseguiu encontrar a solução ótima de todos os problemas com ótimos conhecidos e conseguiu, em média, resultados melhores do que o ILS-DVLNS-cn na grande maioria dos problemas sem ótimos conhecidos e também conseguiu achar a melhor solução na maioria desses problemas.

Por fim, pode-se dizer que os objetivos deste trabalho foram atingidos, com todos os algoritmos implementados e testados. Também se observa que para o conjunto de dados e problemas utilizados o ILS-DVLNS-m se saiu melhor do que os outros métodos de busca aqui estudados.

Como possibilidade de trabalho futuros, destaca-se a realização de testes exaustivos. Esses testes abrangeriam uma quantidade maior de problemas que devem ser executados um maior número de vezes e com um tempo maior de processador para cada execução. Outra possibilidade a ser estudada é uma nova implementação do ciclo negativo, permitindo com que ele seja executado de forma mais rápida e eficiente. A longo prazo, pode ser estudado o uso e aplicação de métodos de Inteligência Artificial, como aprendizado por reforço, em problemas de otimização.

REFERÊNCIAS

AHUJA, R. K.; ERGUN, Ö.; ORLIN, J. B.; PUNNEN, A. P. **A survey of very large-scale neighborhood search techniques.** Discrete Applied Mathematics, v. *123*(1-3), p. 75–102, 2002.

ANGHINOLFI, D.; PAOLUCCI, M. **Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach**. Computers & Operations Research, v. 34(11), p. 3471-3490, 2007.

ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H. **Pesquisa Operacional**. Rio de Janeiro: Elsevier Brasil. 2011.

BAKER, K. R. **Sequencing rules and due-date assignments in a job shop**. Management Science, v. 30(9), p. 1093–1104, 1984.

BEASLEY, J. E. **OR-Library**. Brunel University. Disponível em: <http://people.brunel.ac.uk/~mastjjb/jeb/info.html>. Acesso em 22 dez. 2016.

BIGRAS, L. P.; GAMACHE, M.; SAVARD, G. **Time-indexed formulations and the total weighted tardiness problem**. INFORMS Journal on Computing, v. 20(1), p. 133-142, 2008.

BILGE, Ü.; KIRAÇ, F.; KURTULAN, M.; PEKGÜN, P. **A tabu search algorithm for parallel machine total tardiness problem.** Computers & Operations Research, v. 31(3), p. 397-414, 2004.

BLAZEWICZ, J.; DRABOWSKI, M.; WEGLARZ, J. (1986). **Scheduling Multiprocessor Tasks to Minimize Schedule Length.** IEEE Transactions on Computers, v. 35, p. 389-393, 1986.

BONDY, A.; MURTY, U. S. R. **Graph theory with applications**. Nova Iorque: Elsevier Science Publishing Co, 1976.

BRUCKER, P.; LENSTRA, J. K.; RINNOOY KAN, A. H. G. **Complexity of machine scheduling problems**. Annals of Discrete Mathematics, v. 1, p. 343–362, 1977.

CHENG, BA-YI; LI, KAI. **An Agent Based Heuristic Algorithm for Uniform Parallel Machine Scheduling Problem with Tails**. 2010 International Symposium on Computational Intelligence and Design, Hangzhou, p. 38-42, 2010.

CONGRAM, R. K.; POTTS, C. N.; VAN DE VELDE, S. L. **An iterated Dynasearch algorithm for the single-machine total weighted tardiness scheduling problem**. INFORMS Journal on Computing, v. 14(1), p. 52-67, 2012.

DELLA CROCE, F.; GARAIX, T.; GROSSO, A. **Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem**. Computers & Operations Research, v. 39(6), p. 1213–1217, 2012.

DELLA CROCE, F.; GROSSO, A.; TADEI, R. **An enhanced Dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem**. Operations Research Letters, v. 32(1), p. 68-72, 2004.

DU, J.; LEUNG, J. Y. T. **Minimizing total tardiness on one machine is NP-hard**. Mathematics of Operations Research, v. 15(3), p. 483-495, 1990.

EDIS, E.; OGUZ, C.; OZKARAHAN, I. **Parallel machine scheduling with additional resources: Notation, classification, models and solution methods**. European Journal of Operational Research. v. 230, p. 449–463, 2013.

ERGUN, Ö.; ORLIN, J. B. **Fast neighborhood search for the single machine total weighted tardiness problem**. Operations Research Letters, v. 34(1), p. 41-45, 2006.

FERREIRA, O. E.; SILVA, G. P. **O uso de técnicas de busca em vizinhança de grande porte para o Problema de Sequenciamento de Tarefas em Máquinas Paralelas**. XLIX Simpósio Brasileiro de Pesquisa Operacional, 2017. Disponível em: < http://www.sbpo2017.iltc.br/pdf/168627.pdf>. Acesso em 22 nov. 2018.

FERREIRA, O. E.; SILVA, G. P. **Uma Busca em Vizinhança de Grande Porte para o Sequenciamento de Tarefas em Máquinas Paralelas**. XIX Latin-Iberoamerican Conference on Operations Research, 2018.

FRANCO, R. R.; SILVA, G. P. **A metaheuristic Iterated Local Search and Very Large-scale Neighborhood Search to solve the Parallel Machine Scheduling Problem**. Proceedings of the XVIII Latin-Iberoamerican Conference on Operations Research, v. 1, p. 327-334, 2016.

GALLAGHER, M.; YUAN, B. The Web Page of Max-Set of Gaussians Landscape Generator. Disponível em: < http://boyuan.global-optimization.com/LGMVG/index.htm>. Acesso em 19 jan. 2018.

GRAHAM, R. L.; LAWLER, E. L.; LENSTRA, J. K.; KAN, A. R. **Optimization and approximation in deterministic sequencing and scheduling: a survey**. Annals of discrete mathematics, v. 5, p. 287-326, 1979.

KAO, C.; LEE, H. T. **Discrete time parallel-machine scheduling: a case of ship scheduling**. Engineering Optimization Journal, v. 26, p. 287-294, 2007. Disponível em: < https://doi.org/10.1080/03052159608941123>. Acesso em 22 dez. 2017.

KOULAMAS, C. **Decomposition and hybrid simulated annealing heuristics for the parallel-machine total tardiness problem**. Naval Research Logistics, v. 44, p. 109–25, 1997.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. **Iterated local search**. In Handbook of metaheuristics, Springer US, p. 320-353, 2003.

PAN, Y.; SHI, L. **On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems**. Mathematical Programming, v. 110(3), p. 543-559, 2007.

RODRIGUES, R.; PESSOA, A.; UCHOA, E.; POGGI DE ARAGÃO, M. **Heuristic algorithm for the parallel machine total weighted tardiness scheduling problem**. Relatório de Pesquisa em Engenharia de Produção, v. 8, p. 10, 2008.

Certifico que o aluno **Eduardo de Oliveira Ferreira,** autor do trabalho de conclusão de curso intitulado "**O uso de técnicas de busca em vizinhança de grande porte para resolver o problema de sequenciamento de tarefas em máquinas paralelas e uniformes**", efetuou as correções sugeridas pela banca examinadora e que estou de acordo com a versão final do trabalho.

Gustavo Peixoto Silva - Orientador

Ouro Preto, 26 de novembro de 2018.