# O USO DE BUSCAS EM VIZINHANÇA DE GRANDE PORTE PARA A PROGRAMAÇÃO DE MÁQUINAS PARALELAS
# THE USE OF VERY LARGE-SCALE NEIGHBORHOORD SEARCHS TO SOLVE THE PARALLEL MACHINE SCHEDULING PROBLEM

**Resumo**

Este trabalho trata do problema de sequenciamento de tarefas em máquinas paralelas com atraso total ponderado (*parallel machines total weighted tardiness problem*). O objetivo é sequenciar as tarefas nas máquinas minimizando a soma dos atrasos ponderados. O problema é resolvido em duas etapas: o particionamento das tarefas entre as máquinas e o sequenciamento das tarefas nas máquinas. A contribuição deste trabalho é resolver as duas etapas com diferentes heurísticas de busca de grande porte. A técnica *Very Large-scale Neighborhood Search*, é empregada, de duas formas distintas, para realizar o particionamento das tarefas, e a *Dynasearch* realiza o sequenciamento das tarefas nas máquinas. Ambas as buscas são combinadas na metaheurística ILS e os resultados são comparados entre si com problemas *benchmark* da literatura.

*Palavras-chave*: *parallel machines total weighted tardiness problem; busca em vizinhança de grande porte.*

**Abstract**

This work addresses the parallel machines total weighted tardiness problem. The goal is to sequencing a set of tasks on a set of machines while minimizes the total weighted tardiness. This problem is solved in two stages: the partitioning of tasks between machines and the tasks sequence on each machine. The contribution from this work is to solve both stages through very different large-scale neighborhood search heuristics. Two large-scale neighborhood search techniques are employed to perform the tasks partitioning, and the Dynasearch performs the task sequencing on each machines. Both search heuristics are combined in an ILS metaheuristic and tested with benchmark problems available in literature.

*Keywords*: *parallel machines total weighted tardiness problem; very large-scale neighborhood search.*

## 1. Introdução

O problema do sequenciamento de tarefas em máquinas paralelas e uniformes com atraso total ponderado consiste em particionar as tarefas $J = \{1,..., n\}$ entre as máquinas $M =$

{1,..., $m$} e posteriormente sequenciar as tarefas atribuídas a cada máquina, sem preempção. Cada tarefa $j$ tem um dado tempo de processamento $p_j$, uma data de entrega $d_j$ e um custo por dia de atraso $w_j$. Não é exigida a preparação da máquina entre a execução de duas tarefas quaisquer. O atraso de uma tarefa $j$ em relação à sua data de entrega é definido como $T_j = max\{0, C_j - d_j\}$, onde $C_j$ é o dia de finalização da tarefa. O custo de uma solução $\boldsymbol{S}$ é dado pela soma dos atrasos, ponderados pelos respectivos custos por dia de atraso em cada máquina, ou seja: $c(\boldsymbol{S}) = \sum_{j=1}^{m}\sum_{i=1}^{n(j)} w_i T_i$ , onde $n(j)$ corresponde ao número de tarefas na máquina $j$. Este problema é conhecido na literatura como *parallel machines total weighted tardiness problem*, e é denotado por $Pm||\sum w_j T_j$ (DELLA CROCE; GARAIX; GROSSO, 2012).

O $Pm||\sum w_j T_j$ tem grande importância em aplicações industriais, que normalmente exigem a divisão e o ordenamento de tarefas entre máquinas, tal que torne a produção mais rápida e menos onerosa (ARENALES et al., 2011). Este problema é do tipo *NP-Hard* (LENSTRA; KAN; BRUCKER, 1977; DU; LEUNG, 1990), o que significa que não se conhece um algoritmo com complexidade polinomial para resolvê-lo.

A literatura conta com um número limitado de trabalhos que abordam este problema especificamente. Recentemente Mensendiek, Gupta, e Herrmann (2015) apresentaram uma formulação matemática e um algoritmo do tipo *branch-and-bound* para resolver o problema de máquinas paralelas com data fixa de entrega para minimizar o atraso total. Como o problema é do tipo *NP-hard*, os autores propuseram uma metaheurística Busca Tabu baseada em assinalamento e um Algoritmo Genético híbrido que representa uma solução por meio de uma sequência de tarefas. Outros trabalhos relacionados ao problema são o de Anghinolfi e Paolucci (2007) que propõem uma metaheurística que envolve a Busca Tabu, *Simulated Annealing* e *Variable Neighborhood Search* para o $P||\sum T_j$ e Bilge et al. (2004) que apresentam uma Busca Tabu para o $P|s_{ij}|\sum T_j$ considerando o tempo de preparação dependente da sequência de execução das tarefas.

Rodrigues et al. (2008) tratam especificamente do $Pm||\sum w_j T_j$, sendo que sua principal contribuição é a de representar o problema com múltiplas máquinas por uma única sequência de tarefas, o que diminui muito o tempo de processamento. A sequência única é otimizada empregando a metaheurística *Iterated Local Search* - ILS com movimentos de inserção, que remove uma tarefa de uma posição e a reinsere em outra posição, e o movimento de troca de tarefas, que troca a posição de um par de tarefas. A combinação destes movimentos, dita *generalized pairwise interchanges* - GPI, é melhorada por meio de um critério específico de

desempate. A vizinhança é explorada com a estratégia do primeiro vizinho de melhora, e o processo de busca é reiniciado periodicamente.

Embora a vizinhança GPI seja facilmente adaptada ao ambiente de máquinas paralelas, ela não se aplica à busca GPI *Dynasearch*. A *Dynasearch* é uma busca em vizinhança de grande porte para o sequenciamento de tarefas em uma única máquina. Ela foi proposta por Congram, Potts e van de Velde (2002) para o $1||\sum w_j T_j$, e utiliza um algoritmo de Programação Dinâmica para pesquisar uma vizinhança de tamanho exponencial em tempo polinomial. A *Dynasearch* realiza uma série de trocas independentes que levam à melhoria da solução, enquanto as buscas clássicas fazem apenas um movimento de inserção ou troca por iteração. Assim, ela é capaz de realizar a busca em um espaço muito mais amplo do que as técnicas de busca da primeira ou da melhor solução de melhora, em um mesmo tempo de processamento.

Della Croce, Garaix e Grosso (2012) apresentam uma metaheurística ILS que incorpora a busca *Dynasearch* em cada máquina do $Pm||\sum w_j T_j$ e utiliza uma busca em vizinhança de grande porte (*Very Large-scale Neighborhood Search-VLNS*) de tamanho não polinomial em relação ao número de máquinas, proposta por Ahuja et al. (2002). Para cada par de máquinas ($m_1$, $m_2$), é computada a maior diminuição $\Delta_{m1,m2}$ que pode ser obtida no atraso, devido aos movimentos de inserção e troca de tarefas entre as máquinas. Posteriormente é construído o grafo de melhoria $G = (M, E)$, onde $E = \{\{m_1, m_2\}: \Delta_{m1,m2} > 0\}$. Uma solução vizinha é obtida encontrando um emparelhamento máximo (*maximum matching*) em $G$ e executando os movimentos associados às arestas do emparelhamento.

Este trabalho apresenta uma metaheurística ILS baseada no trabalho de Della Croce, Garaix e Grosso (2012), utilizando a busca *Dynasearch* em cada máquina e uma busca VLNS para otimizar o particionamento das tarefas entre as máquinas. Neste trabalho foi implementado o algoritmo proposto por Della Croce, Garaix e Grosso (2012) e também uma segunda versão de busca do tipo VLNS entre máquinas, construindo um grafo de melhoria que, diferentemente de Della Croce, Garaix e Grosso (2012), considera a troca de tarefas entre várias máquinas, podendo inclusive envolver todas as máquinas. Para realizar a busca, é aplicado um algoritmo para detectar ciclos negativos, os quais levem a uma troca cíclica que reduz o custo da solução corrente (AHUJA, ORLIN, SHARMA, 2000). A duas versões do ILS utilizam a busca *Dynasearch* para realizar o sequenciamento das tarefas em cada máquina. As versões foram testadas com instâncias *benchmark* da literatura, mostrando a eficiência da versão proposta.

Este trabalho está dividido da seguinte forma: na Seção 2 são apresentadas as características da metaheurística utilizada, na Seção 3 é apresentado o método de busca local

empregado em cada etapa da resolução do problema. Na Seção 4 é apresentado o algoritmo ILS implementado para resolver o problema. A Seção 5 contém os resultados dos testes computacionais realizados, e na seção 6 são descritas as conclusões do trabalho.

## 2. *Iterated Local Search* - ILS para Resolver o Problema

De acordo com Lourenço, Martin e Stützle (2003), a metaheurística ILS se baseia em um princípio que tem sido usado em várias heurísticas específicas como a heurística iterativa Lin-Kernigham para resolver o problema do Caixeiro Viajante (JOHNSON, 1990) e a Busca Tabu adaptativa para resolver o problema de Assinalamento Quadrática (TALBI; HAFIDI; GEIB, 1998). Primeiro, uma busca local é aplicada a uma solução inicial. Poteriormente, a cada iteração, é realizada uma perturbação na solução ótima local e finalmente, é aplicada uma busca local na solução perturbada. A solução gerada é aceita como a nova solução corrente de acordo com os critérios de aceitação estabelecidos. Este processo se repete até que o critério de parada seja atingido.

São destacados dois pontos principais que caracterizam a ILS: *i*) deve haver uma única cadeia ou sequência de soluções que esteja sendo construída e, *ii*) a busca por melhores soluções ocorre em um espaço definido por uma heurística de busca local qualquer. Na prática, uma busca local tem sido a heurística embutida mais usada, mas na verdade qualquer "otimizador" pode ser usado, seja ele determinístico ou não. Assim, fica claro que a ILS é uma metaheurística altamente dependente do método de busca local (TALBI, 2009). Neste trabalho foram utilizadas duas técnicas de busca em vizinhança de grande porte para compor a ILS que resolve o problema. A seguir são detalhadas as técnicas de busca local implementadas.

## 3. Heurísticas de Busca Local

Um algoritmo, ou heurística de busca local inicia com uma solução viável e a substitui repetidamente por uma solução melhorada até que algum critério de parada seja satisfeito. Os algoritmos de busca local, são considerados ferramentas importantes para resolver problemas de otimização combinatória de forma eficiente (AAARTS e LENSTRA, 1997). Uma questão fundamental numa heurística de busca local é a escolha da estrutura de vizinhança, isto é, a maneira como uma vizinhança é definida. Esta escolha determine, em grande parte, se a busca encontrará soluções de qualidade ou se ficará presa a ótimos locais muito pobres. Em geral, quanto maior a vizinhança, maiores as chances de se obter uma solução de alta qualidade. Por outro lado, uma vizinhança muito ampla requer um tempo elevado para ser pesquisada. Como

geralmente se realizam muitas iterações da heurística de busca na vizinhança com diferentes pontos de partida, uma vizinhança maior não necessariamente produzir uma heurística mais eficaz, a menos que se possa pesquisar a vizinhança maior de uma maneira muito eficiente.

A busca em vizinhança de grande porte, proposta por Ahuja, Orlin e Sharma (2000) e Ahuja et al. (2002) fazem uso da algoritmos de Fluxo em Redes e de Programação Dinâmica para realizar a busca, diminuindo assim o tempo de processamento em cada iteração enquanto considera toda a vizinhança. Estas técnicas de busca em vizinhança de grande porte foram utilizadas para resolver o problema proposto em todas as suas etapas, compondo assim a metaheurística ILS.

Conforme dito anteriormente, o $Pm||\sum w_j T_j$ é composto de duas etapas, sendo *i*) o particionamento das tarefas entre as diferentes máquinas e *ii*) o sequenciamento das tarefas em cada máquina. Na primeira etapa foram aplicadas, neste trabalho, duas técnicas diferentes de busca em vizinhança de grande porte. A segunda etapa foi resolvida com apenas uma técnica de busca, entretanto, esta também é uma busca em vizinhança de grande porte. A seguir são apresentadas cada uma das técnicas implementadas.

### *3.1 Particionamento das Tarefas via VLNS com Ciclos Negativos*

A técnica de busca VLNS (AHUJA et al. 2002), é uma generalização da vizinhança de realocação e troca aos pares, na qual uma solução vizinha é obtida realizando uma troca cíclica ou uma troca em cadeia dos elementos da solução de um problema de particionamento de conjuntos. 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 uma *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, a qual terá, finalmente, uma tarefa realocada para a máquina 1. No caso de uma *troca em cadeia*, a máquina 3 não realoca qualquer tarefa para a máquina 1 e esta terá uma tarefa a menos, assim como a máquina 3 contará com uma tarefa a mais do que havia anteriormente.

Para realizar uma busca eficiente na vizinhança de troca cíclica e em cadeia, é necessário construir um grafo direcionado de uma dada solução $\boldsymbol{S}$, $G(\boldsymbol{S})$ denominado *grafo de melhoria*. Considerando $n$ tarefas e $m$ máquinas, seja $S[j]$ a máquina que contém a tarefa $j$, então, $G(\boldsymbol{S}) = (N, E)$ é uma grafo direcionado com o conjunto de nós $N$ contendo um nó 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 $E$ contém os seguintes tipos de arcos:

1. $(i, j)$ ligando as tarefas $i$ e $j$ que representa a substituição da tarefa $j$ pela $i$ na máquina $S[j]$. O custo deste tipo de arco é dado por $c(\{i\} \cup S[j] \backslash \{j\}) - c(S[j])$;
2. $(i, m_j)$ ligando a tarefa $i$ à máquina $m_j$, $m_j \neq S[i]$, representado a inserção de $i$ em $m_j$ sem que qualquer tarefa seja retirada de $m_j$. O custo do arco é dado por $c(\{i\} \cup m_j) - c(m_j)$;
3. $(t, j)$ do super-nó $t$ para cada tarefa $j$, que considera a retirada da tarefa $j$ da máquina $S[j]$ sem que qualquer tarefa seja inserida em $S[j]$. O custo deste arco é $c(S[j] \backslash \{j\}) - c(S[j])$;
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 em $G(\boldsymbol{S})$ corresponde a uma troca cíclica, ou em cadeia, que melhora o valor da solução $\boldsymbol{S}$. Esta é uma maneira eficiente de realizar uma busca na vizinhança de $\boldsymbol{S}$. Portanto, basta encontrar um ciclo negativo no grafo de melhoria $G(\boldsymbol{S})$ para se obter um vizinho melhor. Neste trabalho foi implementado o algoritmo w*alk to the root* (CHERKASSKY; GOLDBERG, 1999), com complexidade $O(n^2m)$, para encontrar um ciclo negativo em uma rede com $n$ nós e $m$ arcos.

Uma vez encontrado um ciclo negativo, as trocas são realizadas, a solução e o seu respectivo grafo de melhoria são atualizados, e o processo é repetido até que nenhum ciclo negativo seja encontrado no grafo de melhoria. Neste caso a solução corrente é um ótimo local segundo esta vizinhança. Este procedimento foi implementado para realizar o particionamento das tarefas entre as máquinas. Esta busca será denominada aqui VLNS-CN.

### *3.2 Particionamento das Tarefas via VLNS com Matching*

Neste procedimento de busca local foi utilizado um algoritmo de *matching* com peso máximo (*maximum weighted matching*) para pesquisar a vizinhança. Para construir o grafo de melhoria, são realizados movimentos que transferem uma tarefa de uma máquina para outra. Assim, para uma dada solução $\boldsymbol{S}$, seleciona-se um par de máquinas contendo as sequências de tarefas $\sigma_1$ e $\sigma_2$ e são considerados os seguintes movimentos:

a) extrair uma tarefa $j$ de $\sigma_1$ e inseri-la em $\sigma_2$
b) extrair uma tarefa $j$ de $\sigma_2$ e inseri-la em $\sigma_1$
c) extrair as tarefas $i \in \sigma_1, j \in \sigma_2$ e inserir $i$ em $\sigma_2 - \{ j \}$ e $j$ em $\sigma_1 - \{ i \}$

A combinação de todos os movimentos possíveis utilizando a), b) e c) nas sequências $\sigma_1$ e $\sigma_2$ define a vizinhança, cujo tamanho é não-polinomial em relação ao número de

máquinas. Para pesquisar esta vizinhança em tempo polinomial é aplicado um algoritmo de *matching* com peso máximo em um grafo definido da seguinte forma:

1. para cada par de máquinas ($m_1$, $m_2$), calcular a diminuição máxima do atraso $\Delta_{m1,m2}$ obtida ao se realizar os movimentos a), b) e c) nas sequências $\sigma_1$, $\sigma_2$, para todo $i, j \in \{ \sigma_1 \} \cup \{ \sigma_2 \}$.
2. construir o grafo de melhoria $G(M, E)$ onde: $M$ é o conjunto das máquinas e $E=\{\{m_1, m_2\}: \Delta_{m1,m2} > 0\}$
3. a próxima solução vizinha será obtida aplicando o *matching* em $G$ e executando os movimentos relacionados às arestas selecionadas.

É importante ressaltar que a realização dos movimentos a), b) e c) necessita de $O(n^3)$ operações 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).

Como o algoritmo do *matching* não seleciona arestas que podem piorar o custo da solução corrente, é garantido que sempre que o problema possuir um número par de máquinas, poderá ocorrer o emparelhamento perfeito. Também é garantido que para qualquer número de máquinas, com qualquer número de tarefas, ocorrerá o emparelhamento máximo quando não houver emparelhamento perfeito (BONDY; MURTY, 1976). Isso garante que o algoritmo consiga realizar a máxima melhoria possível no grafo gerado pelos passos 1., 2. e 3 por meio dos movimentos a), b) e c), descritos acima. Esta busca será denominada neste trabalho VLNS-M.

### *3.3 Sequenciamento das Tarefas em cada máquina via Dynasearch Swap*

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

Uma forma eficiente de encontrar o melhor conjunto de trocas independentes de uma permutação $\sigma$ consiste em utilizar um algoritmo de Programação Dinâmica. Este algoritmo

adota o esquema de enumeração *forward* no qual as tarefas são adicionadas ao final da sequência parcial e são trocadas de posição com as demais tarefas. Define-se que para uma sequência parcial estar no estado ($k$, $\sigma$), $k$ =1,..., $n$, ela pode ser obtida a partir da sequência parcial ($\sigma(1)$, ..., $\sigma(k)$) aplicando uma série de movimentos independentes. Para encontrar a melhor sequência na vizinhança *Dynasearch Swap* de $\sigma$, que define o estado ($n$, $\sigma$), 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 Swap*.

Seja $\sigma_k$ a sequência parcial com a soma mínima dos atrasos ponderados para as tarefas $\sigma(1)$, ..., $\sigma(k)$ entre as sequências parciais no estado ($k$, $\sigma$). Seja também, $F(\sigma_k)$ a soma dos atrasos ponderados para as tarefas $\sigma(1)$, ..., $\sigma(k)$ em $\sigma_k$. Esta sequência parcial é obtida a partir de uma outra sequência parcial $\sigma_i$ que tem valor objetivo mínimo entre todas sequências parciais em algum estado prévio ($i$, $\sigma$), $0 \leq i < k$, 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 Swap*. O processo é repetido para uma série de soluções iniciais distintas, guardando sempre a melhor solução encontrada.

### *3.4 Algoritmo de Programação Dinâmica para uma única máquina*

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

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

$$
\begin{aligned}
&F(\sigma_0) = 0, \text{ e considere } (x)^+ = \max\{0, x\} \text{ e } P_{\sigma(k)} = \sum_{i=1}^{k} p_{\sigma(i)} \\
&F(\sigma_1) = w_{\sigma(1)}(p_{\sigma(1)} - d_{\sigma(1)})^+ \\
&F(\sigma_k) = \min\left\{ F(\sigma_{k-1}) + w_{\sigma(k)}(P_{\sigma(k)} - d_{\sigma(k)})^+, \right. \\
&\qquad \left. \min_{0 \leq i \leq k-2} \left\{ \begin{array}{l} F(\sigma_i) + w_{\sigma(k)}(P_{\sigma(i)} + p_{\sigma(k)} - d_{\sigma(k)})^+ \\ + \sum_{j=i+2}^{k-1} w_{\sigma(j)}(P_{\sigma(j)} + p_{\sigma(k)} - p_{\sigma(i+1)} - d_{\sigma(j)})^+ + w_{\sigma(i+1)}(P_{\sigma(k)} - d_{\sigma(i+1)})^+ \end{array} \right\} \right\} \quad (1)
\end{aligned}
$$

Assim, o valor da solução ótima é igual a $F(\sigma_n)$, e a sequência pode ser obtida por um processo do tipo *backtracking*. Por exemplo, se o valor de $F(\sigma_k)$ for dado pelo primeiro termo

(linha) na minimização (1), então a tarefa $\sigma(n)$ não é trocada de posição em relação à solução corrente, e prossegue-se para saber como $F(\sigma_{(n-1)})$ foi obtido. Caso contrário, se o valor de $F(\sigma_k)$ for dado pelo segundo termo, para algum índice $i$, então as tarefas das posições $\sigma(n)$ e $\sigma(i+1)$ devem ser trocadas na nova solução, e prossegue-se 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 as trocas são identificadas e o procedimento termina.

*3.5 Algoritmo de busca utilizando a vizinhança Dynasearch Swap*

A implementação do algoritmo com busca na vizinhança *Dynasearch Swap* inicia com uma solução inicial $\sigma^0$ obtida por algum método guloso. Durante a iteração $t$, $\sigma^{t-1}$ é a solução corrente a ser melhorada. Usando o algoritmo de Programação Dinâmica, é possível calcular o valor de $F(\sigma_k^{t-1})$ para $k = 1, ..., n$, e aplica-se um procedimento do tipo *backtracking* para encontrar a correspondente sequência $\sigma^t$. A solução definida por $\sigma^t$ é um ótimo local na vizinhança *Dynasearch Swap* se $F(\sigma_k^{t-1}) = F(\sigma_k^{t-2})$, onde $F(\sigma_k^{-1})$ representa o valor objetivo da solução inicial $\sigma_k^0$. Neste caso, o algoritmo é encerrado. Por outro lado, se $F(\sigma_k^{t-1}) < F(\sigma_k^{t-2})$, então deve ser executada uma nova iteração tendo $\sigma_k^t$ como a solução corrente.

A solução inicial é obtida pelo método *Early Due Date* – EDD (Baker 1984), que consiste em priorizar a alocação das tarefas com menor data de entrega. A tarefa com menor data de entrega é incluída no final da máquina que estiver disponível mais cedo. O processo se repete até que todas as tarefas tenham sido alocadas.

## 4. A Metaheurística ILS para o resolver o $Pm||\sum w_j T_j$

A metaheurística ILS é inicializada com uma solução obtida pelo método EDD na linha 1 do Algoritmo 1, e tem com duas heurísticas de busca local. A **`dynasearch_swap(S, N1)`**, na linha 4, tenta melhorar o custo de cada máquina isoladamente pela vizinhança *Dynasearch Swap*. A busca **`very_large_neighborhood_search(S1, N2)`**, na linha 6, considera movimentos entre máquinas do tipo *n-optimal*, ou seja, utilizando uma das buscas VLNS. Na linha 19 ocorre a perturbação intra-máquina com a função **`kick1(S)`**, que modifica as posições das tarefas em máquinas escolhidas aleatoriamente. Posteriormente, na linha 22, é realizada a perturbação **`kick2(S)`**, que faz movimentos de troca de tarefas escolhidas entre pares de máquinas tomadas aleatoriamente.

```
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_swap(S, N1);
5.      Repita
6.          S2 = very_large_neighborhood_search(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;
26. retona S*;
```

Algoritmo 1. Pseudocódigo da implementação do ILS.

O Algoritmo 1 foi testado em duas versões, sendo que em ambas, a busca local em cada máquina é realizada pelo procedimento *Dynasearch Swap* descrito na seção 3.3. Na primeira versão, denominada ILS1, a busca local em várias máquinas (linha 6 do Algoritmo 1) é realizada pela heurística VLNS-CN. Por outro lado, na segunda versão esta busca é substituída pela heurística VLNS-M, e esta versão é denominada ILS2.

## 5. Testes Computacionais

Foi utilizado um computador com processador Intel (R) Pentium (R) N3540 @ 2.16 GHz e 4 GB de memória RAM para realizar os testes computacionais. Inicialmente foram resolvidos problemas cuja solução ótima é conhecida, verificando a eficiência das versões do ILS. Posteriormente foram utilizados problemas maiores, cujas soluções ótimas não são conhecidas. Os resultados são comparados e analisados ao final.

### *5.1 Comparação entre o ILS1 e o ILS1 para problemas com solução ótima conhecida*

Os resultados apresentados na Tabela 1 são de problemas com 20 tarefas, e o número de máquinas m = 2, 4,..., 10. Cada problema foi resolvido 5 vezes, verificando quantas vezes cada versão atingiu a solução ótima. Os parâmetros R e T, apresentados na Tabela 1, são o *due date range* e o *tardiness fator* respectivamente, que caracterizam e determinam a

dificuldade de cada instância (Crauwels et al. 1998). O tempo de cada execução foi de 15 minutos.

Na Tabela 1, a linha "%ot" corresponde ao percentual da diferença entre a melhor solução obtida e a solução ótima do problema. A linha "QT" contém o número de vezes que a versão atingiu a solução ótima nas 5 rodadas. Tomando como exemplo $m = 2$, a versão ILS1 e o problema número 8, este foi gerado para R = 0,6, T = 0,6 e o % da diferença entre a melhor solução obtida e a solução ótima foi igual a 0,45%. Neste caso, o ILS1 não alcançou a solução ótima em nenhuma das 5 rodadas, conforme o valor zero na linha QT. Por outro lado, a versão ILS2 alcançou a solução ótima nas 5 rodadas, obtendo %ot igual e 0 e QT igual a 5.

Para estes problemas, o ILS1 obteve a solução ótima na maioria dos problemas com 2 e 4 máquinas e em todos os problemas com 6 máquinas. Contudo, alcançou o ótimo em apenas 6 dos 15 problemas com 8 máquinas e em apenas 8 dos 15 problemas com 10 máquinas. Este resultado pode indicar que uma quantidade muito baixa de tarefas por máquinas compromete o desempenho da ILS1. Já a versão ILS2 obteve a solução ótima em todos os problemas testados em todas as 5 execuções de cada problema. Como as duas metaheurísticas utilizam o *Dynasearch*, percebe-se que o desempenho do ILS2 se deve ao procedimento de busca entre máquinas, que nesta versão utiliza o VLNS-M.

Tabela 1 - Resultados obtidos utilizando ILS1 e ILS2 para problemas com n = 20 e m = 2, 4, 6, 8 e 10 cujas soluções ótimas são conhecidas.

| Problema | | # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **R** | 0,2 | 0,2 | 0,2 | 0,4 | 0,4 | 0,4 | 0,6 | 0,6 | 0,6 | 0,8 | 0,8 | 0,8 | 1 | 1 | 1 |
| | | **T** | 0,2 | 0,6 | 1 | 0,2 | 0,6 | 1 | 0,2 | 0,6 | 1 | 0,2 | 0,6 | 1 | 0,2 | 0,6 | 1 |
| ***m* = 2** | **ILS1** | **%ot** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0,45 | 0 | 0 | 0 | 0,15 | 0,04 | 0 | 0 |
| | | **QT** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 0 | 5 | 5 | 5 | 0 | 0 | 5 | 5 |
| | **ILS2** | **%ot** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **QT** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| ***m* = 4** | **ILS1** | **%ot** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0,53 | 0 | 0 |
| | | **QT** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 0 | 5 | 5 |
| | **ILS2** | **%ot** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **QT** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| ***m* = 6** | **ILS1** | **%ot** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **QT** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| | **ILS2** | **%ot** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **QT** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| ***m* = 8** | **ILS1** | **%ot** | 0 | 0 | 0 | 0 | 0,05 | 0 | 2,2 | 0,12 | 0,27 | 0,55 | 0,43 | 0,11 | 0 | 0,5 | 0,43 |
| | | **QT** | 5 | 5 | 5 | 5 | 4 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 |
| | **ILS2** | **%ot** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **QT** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| ***m* = 10** | **ILS1** | **%ot** | 2,56 | 0 | 0 | 0 | 0 | 0 | 0,26 | 0 | 0 | 0,69 | 1,30 | 0,11 | 0 | 0,63 | 1,30 |
| | | **QT** | 0 | 5 | 5 | 5 | 5 | 5 | 0 | 5 | 5 | 0 | 0 | 0 | 5 | 0 | 0 |
| | **ILS2** | **%ot** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **QT** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |

Na Tabela 1, pode-se observar que o número de soluções ótimas encontradas pelo ILS1 aumentou quando *m* aumento de 2 para 4 e depois para 6. Posteriormente, houve uma queda acentuada quando *m* passou para 8 e uma leve recuperação para *m* = 10. Desta forma, há evidências que o ILS1 obtém seus melhores resultados nos problemas em que cada máquina deve realizar um grande número de tarefas. Resta destacar que, embora o ILS1 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 0,98% da solução ótima para *m* = 10. Já o ILS2 se mostra mais eficiente em todos os cenários, independentemente da quantidade de tarefas em cada máquina.

*5.2 Comparação entre o ILS1 e o ILS2 para problemas sem solução ótima conhecida*

As versões ILS1 e ILS2 também foram testadas para problemas cujas soluções ótimas não são conhecidas. Os dez problemas contam com 50 tarefas e o número de máquinas variando entre 2 e 10, ou seja, *m* = 2, 4, ..., 10. Na Tabela 2 são apresentadas as melhores soluções encontradas para cada problema (nas colunas), e para cada versão (nas linhas), após 10 execuções, com duração de 15 minutos cada uma. A linha "Min" contém o valor da função objetivo, a linha "DP" corresponde ao desvio padrão das 10 soluções e na linha "%Dif" é apresentada a diferença percentual do valor da FO das duas versões, ou seja, %Dif = (ILS1 – ILS2)/ILS2. Esta informação mostra que, embora a versão ILS2 obtenha soluções melhores do que o ILS1 em várias instâncias, esta superioridade é muito pequena em termos absolutos e percentuais. Os valores destacados em verde correspondem aos melhores resultados e aqueles em rosa, aos piores resultados. Nos problemas sem destaque houve empate entre as duas versões. A coluna S* mostra o total de vitórias de cada versão.

Tabela 2 - Resultados obtidos utilizando ILS1 e ILS2 para problemas com n = 50 e m = 2, 4, 6, 8 e 10 cujas soluções ótimas não são conhecidas.

| | **Prob** | # | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | S* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***m* = 2** | **ILS1** | **Min** | 5795 | 37999 | 30 | 14339 | 103136 | 3372 | 40135 | 0 | 9026 | 62996 | 1 |
| | | **DP** | 0,00 | 2,51 | 0,00 | 2,68 | 0,45 | 0,00 | 6,88 | 0,00 | 26,43 | 4,55 | |
| | **ILS2** | **Min** | 5795 | 37999 | 30 | 14339 | 103125 | 3372 | 40120 | 0 | 9028 | 62992 | 3 |
| | | **DP** | 0,00 | 1,60 | 0,00 | 0,00 | 1,60 | 0,00 | 7,40 | 0,00 | 26,78 | 5,04 | |
| | **%Dif** | | 0,0000 | 0,0000 | 0,0000 | 0,0000 | 0,0001 | 0,0000 | 0,0004 | 0,0000 | -0,0002 | 0,0001 | |
| ***m* = 4** | **ILS1** | **Min** | 3434 | 20992 | 59 | 8423 | 55755 | 2287 | 22984 | 0 | 6315 | 34433 | 2 |
| | | **DP** | 0 | 2,77 | 0 | 3,85 | 2,3 | 0,89 | 17,22 | 0 | 10,01 | 10,55 | |
| | **ILS2** | **Min** | 3434 | 20985 | 59 | 8420 | 55757 | 2287 | 22977 | 0 | 6322 | 34432 | 4 |
| | | **DP** | 0 | 3,37 | 0 | 3,26 | 1,2 | 0 | 16,29 | 0 | 10,42 | 4,71 | |
| | **%Dif** | | 0,0000 | 0,0003 | 0,0000 | 0,0004 | 0,0000 | 0,0000 | 0,0003 | 0,0000 | -0,0011 | 0,0000 | |
| ***m* = 6** | **ILS1** | **Min** | 4692 | 17736 | 1232 | 9752 | 41081 | 4794 | 20217 | 507 | 8767 | 26492 | 2 |
| | | **DP** | 0,84 | 3,05 | 0,89 | 10,67 | 1,52 | 1,48 | 3,13 | 2,92 | 5,94 | 2,41 | |
| | **ILS2** | **Min** | 4692 | 17734 | 1232 | 9739 | 41080 | 4790 | 20221 | 507 | 8754 | 26493 | 5 |
| | | **DP** | 0,00 | 2,40 | 0,00 | 6,76 | 2,71 | 2,28 | 5,31 | 1,26 | 9,48 | 1,60 | |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %Dif | | 0,0000 | 0,0001 | 0,0000 | 0,0013 | 0,0000 | 0,0008 | -0,0002 | 0,0000 | 0,0015 | 0,0000 | |
| $m = 8$ | ILS1 | Min | 2276 | 12972 | 115 | 5783 | 32251 | 1800 | 14713 | 0 | 5272 | 20488 | 0 |
| | | DP | 1,30 | 5,34 | 1,10 | 6,91 | 1,64 | 8,51 | 5,86 | 0,00 | 7,57 | 2,30 | |
| | ILS2 | Min | 2269 | 12971 | 110 | 5769 | 32248 | 1792 | 14696 | 0 | 5254 | 20483 | 9 |
| | | DP | 2,24 | 1,33 | 1,41 | 2,79 | 0,89 | 5,42 | 8,52 | 0,00 | 5,46 | 2,24 | |
| | %Dif | | 0,0031 | 0,0001 | 0,0455 | 0,0024 | 0,0001 | 0,0045 | 0,0012 | 0,0000 | 0,0034 | 0,0002 | |
| $m = 10$ | ILS1 | Min | 2113 | 11369 | 165 | 5635 | 27568 | 1908 | 13162 | 0 | 5386 | 17807 | 0 |
| | | DP | 3,21 | 0,84 | 0,84 | 7,54 | 1,1 | 10,08 | 3,51 | 0 | 6,3 | 2,17 | |
| | ILS2 | Min | 2112 | 11361 | 165 | 5624 | 27568 | 1876 | 13162 | 0 | 5378 | 17805 | 6 |
| | | DP | 2,28 | 2,23 | 0 | 1,9 | 1,6 | 15,56 | 3,83 | 0 | 6,37 | 2,28 | |
| | %Dif | | 0,0005 | 0,0007 | 0,0000 | 0,0020 | 0,0000 | 0,0171 | 0,0000 | 0,0000 | 0,0015 | 0,0001 | |

É possível verificar que a versão ILS2 foi mais eficiente na resolução do problema com 50 tarefas para qualquer quantidade de máquinas e que o desempenho da versão ILS1 diminui na medida em que o número de máquinas aumenta. Este é outro indício de que a diminuição do número de tarefas por máquina prejudica a eficiência da ILS1. Por outro lado, ainda que a versão ILS1 não produza as melhores soluções, seus resultados são extremamente competitivos, uma vez que na grande maioria dos problemas testados, a diferença percentual é da ordem de $10^{-3}$ a $10^{-4}$, em relação aos resultados obtidos pela ILS2.

## 6. Conclusões

Neste trabalho foram apresentadas duas versões da metaheurítica ILS que utilizam busca em vizinhança de grande porte nas duas etapas de otimização do problema, ou seja, no particionamento das tarefas entre as máquinas e no sequenciamento destas tarefas em cada máquina. A primeira versão é uma implementação inédita para o problema, que utiliza um grafo de melhoria que permite a realocação em cadeia das tarefas entre as máquinas. Neste caso a busca é feita por um algoritmo de detecção de ciclos negativos. A segunda versão, a qual se mostrou mais eficiente, é a um algoritmo proposto na literatura que utiliza um algoritmo de *matching* para realizar o particionamento das tarefas entre as máquinas.

Embora a versão proposta não tenha sido a mais eficiente, é possível verificar a sua competitividade tendo em vista a pequena diferença nas soluções obtidas. Desta forma, este trabalho abre uma frente de pesquisa tendo como continuidade o aprimoramento da versão proposta, que se mostra bastante promissora.

## Referências

AARTS, E., LENSTRA, J.K., 1997. **Local Search in Combinatorial Optimization**. Wiley, New York.

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

AHUJA, R. K.; ORLIN, J. B.; SHARMA, D. Very large-scale neighborhood search. **International Transactions in Operational Research**, v. 7, n. 4-5, p. 301-317, 2000.

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

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. n. 9, p. 1093–1104, 1984.

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

BONDY, J. A.; MURTY, U. S. R. **Graph theory with applications**, v. 290. London: Macmillan, 1976.

CHERKASSKY, B. V.; GOLDBERG, A. V. Negative-cycle detection algorithms. **Mathematical Programming**, v. 85, n. 2, p. 277-311, 1999.

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, n. 1, p. 52-67, 2002.

CRAUWELS, H. A. J.; POTTS, C. N.; VAN WASSENHOVE, L. N. Local search heuristics for the single machine total weighted tardiness scheduling problem. INFORMS Journal on computing, v. 10. n. 3, p. 341-350, 1998.

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, n. 6, p. 1213–1217, 2012.

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

JOHNSON, D. S. Local optimization and the traveling salesman problem. In: **International Colloquium on Automata, Languages, and Programming**. Springer, Berlin, Heidelberg, p. 446-461, 1990.

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

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: GLOVER, F.; KOCHENBERGER, G. A. (Eds.) **Handbook of Metaheuristics**. Dordrecht: Kluwer Academic Publishers, 2003. cap. 11, p. 321-353.

MENSENDIEK, A.; GUPTA, J. N.; HERRMANN, J. Scheduling identical parallel machines with fixed delivery dates to minimize total tardiness. **European Journal of Operational Research**, v. 243, n. 2, p. 514-522, 2015.

RODRIGUES, R.; PESSOA, A.; UCHOA, E.; DE ARAGÃO, M. P. **Heuristic algorithm for the parallel machine total weighted tardiness scheduling problem**. Relatório de Pesquisa em Engenharia de Produção, v. 8, n. 10, Universidade Federal Fluminense, Niterói, RJ, 2008.

TALBI, E. G. **Metaheuristics: from design to implementation**. John Wiley & Sons, 2009.

TALBI, E. G.; HAFIDI, Z.; GEIB, J. M. A parallel adaptive tabu search approach. **Parallel computing**, v. 24, n. 14, p. 2003-2019, 1998.