Recebido: 29/05/2018
Aprovado: 08/01/2019

O uso de técnicas de busca em vizinhança de grande porte para o problema de programação de máquinas paralelas

The use of very large-scale neighbourhood search techniques for the parallel machine scheduling problem

Eduardo de Oliveira Ferreira[1] - Universidade Federal de Ouro Preto – Dep. de Engenharia de Controle e Automação
Gustavo Peixoto Silva[2] - Universidade Federal de Ouro Preto – Dep. de Computação

RESUMO

Este trabalho trata do problema de sequenciamento de tarefas em máquinas paralelas e uniformes (*parallel machines total weighted tardiness problem*). O objetivo é sequenciar as tarefas tal que cada tarefa seja realizada em uma máquina, cada máquina realize uma tarefa por vez e seja minimizada 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 consiste em resolver as duas etapas com diferentes heurísticas de busca de grande porte e comparar suas eficiências. A técnica *Very Large-scale Neighborhood Search*, que utiliza um grafo de melhoria com cancelamento de ciclos negativos, é empregada para realizar o particionamento das tarefas. Um algoritmo de Programação Dinâmica, conhecido como *Dynasearch*, que também é uma técnica de busca em vizinhança de grande porte, realiza o sequenciamento das tarefas em cada máquina. Ambas as buscas são combinadas na metaheurística ILS. As duas versões do ILS são comparadas, resolvendo problemas *benchmark* da literatura.
Palavras-chave: Parallel machines total weighted tardiness problem. Busca em vizinhança de grande porte. Programação dinâmica.

Editor Responsável: Prof. Dr. Hermes Moretti Ribeiro da Silva

ABSTRACT

This study addresses the parallel machine's total weighted tardiness problem. The goal is to sequence a set of tasks on a set of machines so that each task is performed by a single machine; each machine executes a single task at a time while minimizing the total weighted tardiness. There are two fundamental questions to be addressed: the partitioning of tasks between machines and the task's sequence on each machine. The contribution from this work is to solve both steps through very large-scale neighborhood search heuristics. The large-scale neighborhood search technique, which employs improvement graphs with negative cycle cancelling, is adopted to perform the partitioning of tasks. A Dynamic Programming algorithm, called Dynasearch, performs task sequencing on each machine. Both search heuristics are combined in an ILS metaheuristic and the results are compared by solving benchmarking problems from the literature.
Keywords: *Parallel machines total weighted tardiness problem. Very large-scale neighborhood search. Dynamic programming.*

1. deoliveira.eng@gmail.com; 2. Campus Universitário, Morro do Cruzeiro, 35.400-000, Ouro Preto/MG, gustavo@ufop.edu.br

FERREIRA, E.O.; SILVA, G.P. O uso de técnicas de busca em vizinhança de grande porte para o problema de programação de máquinas paralelas. **GEPROS. Gestão da Produção, Operações e Sistemas**, v. 14, n. 5, p. 48 - 66, 2019.

DOI: 10.15675/gepros.v14i5.2370

1. INTRODUÇÃO

Os métodos de solução para o problema de sequenciamento de tarefas em máquinas paralelas têm sido amplamente estudados desde a década de 1950 (GRAHAM *et al.*, 1979). Desde então, muitos algoritmos para a solução dessa classe de problemas têm sido propostos (AHUJA *et al.*, 2002). O sequenciamento de tarefas é um problema comum, em que as tarefas devem ser inicialmente particionadas entre as máquinas e posteriormente sequenciadas em cada uma das máquinas. Neste trabalho foi abordado o problema de sequenciamento de tarefas em máquinas paralelas e idênticas com a penalização por atraso, ou seja, o *parallel-machines total weighted tardiness problem*, denotado por $Pm||\sum w_j T_j$ (DELLA CROCE**;** GARAIX; GROSSO, 2012).

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, já que as aplicações industriais, em grande parte, exigem que o sequenciamento das tarefas seja realizado de forma que elas fiquem divididas entre máquinas e ordenadas dentro de cada uma delas, tornando a produção mais rápida e menos onerosa (ARENALES *et al.*, 2011). O problema aqui estudado é, segundo Lenstra, Kan e Brucker, (1977) e Du e Leung (1990), do tipo *NP-Hard*. Isso significa que não se conhece algoritmo de complexidade polinomial que consiga resolver o problema em um tempo razoável de processamento. Desta forma, as metaheurísticas são muito utilizadas para resolver problemas de grande porte desta natureza.

Um dos problemas enfrentados na implementação de heurísticas de busca local, 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 (AHUJA *et al.*, 2002) tanto para particionar as tarefas entre as máquinas, quanto para definir a sequencia das tarefas de cada máquina. As duas etapas da resolução do problema foram integrada na metaheurística *Iterated Local Search* – ILS (LOURENÇO; MARTIN; STÜTZLE, 2003), como proposto por Della Croce, Garaix e Grosso, (2012).

O diferencial deste trabalho está na aplicação de uma busca VLNS, ainda não explorada para este problema, para realizar o particionamento das tarefas. Esta busca gera um grafo de melhoria próprio e faz uma busca na vizinhança por meio de cancelamento de ciclos negativos, que levam a melhorias no valor da função objetivo. Para verificar a eficiência desta busca, foi implementado também o algoritmo proposto por Della Croce, Garaix e Grosso, (2012), que realiza uma outra busca do tipo VLNS para esta mesma etapa do problema. Della Croce, Garaix e Grosso, (2012) geram um outro grafo de melhoria e utiliza um algoritmo de *matching* com peso máximo para explorar a vizinhança. O sequenciamento das tarefas em cada máquina também é otimizado com uma busca VLNS, no caso a *Dynasearch*, que emprega um algoritmo de Programação Dinâmica para realizar várias trocas de tarefas por iteração, o que impede que a busca seja atraída para ótimos locais pobres e faz com que ela consiga encontrar ótimos melhores do que os métodos tradicionais (CONGRAM; POTTS; VAN DE VELDE, 2002).

Neste trabalho pretende-se verificar a eficiência de uma combinação de técnicas de otimização que ainda não foi empregada no problema. São apresentados os resultados de um experimento computacional, comparando o desempenho da versão proposta com o desempenho da versão da literatura, tendo como referência o valor total do atraso ponderado, ou seja, baseado em informações quantitativas. Para realizar os testes computacionais, foram utilizados dados de problemas *benchmark* disponíveis na literatura.

Este trabalho está dividido da seguinte forma: na Seção 2 são apresentados os elementos teóricos e o método de resolução do problema, na Seção 3 é apresentada a metodologia da pesquisa conduzida neste trabalho. Na Seção 4 são apresentados os resultados dos experimentos computacionais e a Seção 5 contém as conclusões do trabalho.

2. MÉTODO DE RESOLUÇÃO DO PROBLEMA

Para problemas do tipo $Pm||\sum w_j T_j$, é dado um conjunto de tarefas $N = \{1, 2, ..., n\}$, com os tempos de processamento p_j, os pesos w_j além das datas de entrega d_j especificadas para cada tarefa $j \in N$. As tarefas são processadas em uma sequência S em um conjunto $M = \{1, 2, ..., m\}$ de máquinas idênticas 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 por (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, Gamache e Savard, (2008), Pan e Shi (2007), Rodrigues *et al*., (2008), Congram, Potts e Van de Velde, (2002) e Grosso, Della Croce e Tadei, (2004). No entanto, a literatura é razoavelmente limitada para o problema de máquinas paralelas. Os trabalhos mais significativos e que serviram de referências para o estudo foram os de Anghinolfi e Paolucci (2007), Bilge *et al*., (2004), Koulamas, (1997), Rodrigues *et al*., (2008) e Della Croce, Garaix e Grosso (2012).

A seguir são apresentados os procedimentos adotados para realizar o particionamento das tarefas entre as máquinas e aquele empregado para sequenciar as tarefas em cada máquina, lembrado que todos os procedimentos apresentados fazem uso de técnicas de busca em vizinhança de grande porte.

2.1 Particionamento das tarefas entre as máquinas via VLNS com ciclos negativos (VLNS-CN)

Técnicas de busca local do tipo *2-optimal* são muito utilizadas para resolver problemas de otimização combinatória, entretanto, existem vizinhanças mais amplas que permitem 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 a mais 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, $G(\boldsymbol{S}) = (\boldsymbol{V}, \boldsymbol{E})$ é uma grafo direcionado com o conjunto de nós $\boldsymbol{V}$ contendo um nó para cada tarefa $j = 1, .., n$, 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 no grafo $G(\boldsymbol{S})$ corresponde a uma troca cíclica 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 $\boldsymbol{S}$ 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; GOLDBERG, 1999), que tem complexidade $O(n^2m)$ para encontrar um ciclo negativo em $G(\boldsymbol{S})$.

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 nenhum ciclo negativo seja encontrado no grafo de melhoria. Quando o grafo não apresentar qualquer ciclo negativo, então a solução corrente será uma solução ótima local segundo esta vizinhança. Como

esta busca se caracteriza por utilizar um algoritmo para detectar ciclos negativos, ela será denominada doravante por VNLS-CN.

2.2 Particionamento das tarefas entre máquinas via VLNS com Matching (VLNS-M)

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 ***S***, para cada par de máquinas contendo as sequências de tarefas σ_1 e σ_2, são considerados 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 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 σ_1 e σ_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 de melhoria 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 σ_1, σ_2 de m_1 e m_2 respectivamente.
2. construir o grafo de melhoria $G(M, E)$ onde M é o conjunto das máquinas e $E=\{\{m_1, m_2\} \mid \Delta_{m1,m2} > 0\}$
3. a próxima solução vizinha será obtida aplicando o *matching* em G e executando os movimentos relacionados às arestas selecionadas pelo algoritmo.

É importante ressaltar que a realização dos movimentos a), b) e c) tem complexidade $O(n^3)$ e portanto, 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 contar com 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 consegue 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.

2.3 Vizinhança Dynasearch Swap para uma única máquina

Para um problema de sequenciamento de tarefas em uma máquina, o *Swap* é uma vizinhança *2-optimal* que troca duas tarefas quaisquer independentemente delas 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 do ponto de vista computacional. 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 realizada na posição i, para $i = 1,..., n$. A vizinhança *Swap* de uma permutação σ compreende a 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 Swap* 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 Swap* consiste de todas as soluções que podem ser obtidas a partir de σ por uma série 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, de maneira eficiente, é 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 (σ (1), ..., σ (k)) aplicando uma série de movimentos independentes. Para encontrar a melhor sequência na vizinhança *Dynasearch Swap* 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 Swap*.

Seja σ_k a sequência parcial com a soma mínima dos atrasos ponderados para as tarefas σ (1), ..., σ (k) entre todas as sequências parciais no estado (k, σ). Além disto, seja $F(\sigma_k)$ a soma dos atrasos ponderados para as tarefas σ (1), ..., σ (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 σ (k) no final da sequencia se $i = k - 1$, ou adicionando primeiro as tarefas σ (i + 1), ..., σ (k) e então trocando de posição as tarefas σ (i + 1) e σ (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.

2.4 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 Swap*. 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 σ (1), ..., σ (k) em σ_k. Assim, temos a inicialização dada pelas equações (2) e (3) a seguir:

$$F(\sigma_0) = 0 \tag{2}$$

$$F(\sigma_1) = w_{\sigma(1)}\left(p_{\sigma(1)} - d_{\sigma(1)}\right)^+ \tag{3}$$

Onde $(x)^+ = max\{0, x\}$. Então, a recursão para k = 2, ..., n é dada pela equação (4):

$$F(\sigma_k) = min \begin{cases} F(\sigma_{k-1}) + w_{\sigma(k)}\left(P_{\sigma(k)} - d_{\sigma(k)}\right)^+, \\ \min\limits_{0 \le i \le k-2} \begin{cases} F(\sigma_i) + w_{\sigma(k)}\left(P_{\sigma(i)} + p_{\sigma(k)} - d_{\sigma(k)}\right)^+ \\ + \sum_{j=i+2}^{k-1} w_{\sigma(j)}\left(P_{\sigma(j)} + p_{\sigma(k)} - p_{\sigma(i+1)} - d_{\sigma(j)}\right)^+ \\ + w_{\sigma(i+1)}\left(P_{\sigma(k)} - d_{\sigma(i+1)}\right)^+ \end{cases} \end{cases} \quad (4)$$

O valor da solução ótima obtida pelo algoritmo de Programação Dinâmica 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 então todos os *swaps* independentes são identificados e o procedimento de *backtracking* termina.

2.5 O Algoritmo de busca utilizando a vizinhança Dynasearch Swap

A implementação do algoritmo com busca na vizinhança *Dynasearch Swap* inicia com uma solução σ^0 obtida por algum método guloso. Durante a iteração *t*, σ^{t-1} é a solução corrente, que se pretende melhorar realizando uma busca na vizinhança *Dynasearch Swap*. Usando o algoritmo de Programação Dinâmica, apresentado na seção anterior, é possível calcular o valor de F (σ_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 Swap* 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 $F(\sigma_n^{t-1}) < F(\sigma_n^{t-2})$ então deve ser executada uma nova iteração tendo σ^t como a solução corrente.

2.6 Geração da Solução Inicial

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

2.7 A Metaheurística ILS para o Problema $Pm||\sum w_j T_j$

A metaheurística ILS é inicializada com a solução gerada pelo método *EDD*() na linha 1 do Algoritmo 1, e conta com as duas heurísticas de busca local. A **dynasearch_swap(S, N_1)**, na linha 4, tenta melhorar o custo de cada máquina isoladamente pela vizinhança *Dynasearch Swap*. A busca **very_large_neighborhood_search**(S_1, N_2), 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.

Figura 1 - Algoritmo 1. Pseudocódigo da implementação do ILS.

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_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.      iterCont = 0;
14.    senão
15.      iterCont = iterCont + 1;
16.    fim_se;
17.    se N2 falhar em melhorar S1 então
```

```
18.          Se iterCont > MaxSemMelhora então
19.             S = kick1(S*);
20.             iterCont = 0;
21.          senão
22.             S = kick2(S2);
23.          fim_se;
24.       fim_se;
25.    fim_enquanto;
26.    retorna S*;
```

Fonte: Os autores.

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 2.3. 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.

3. METODOLOGIA DA PESQUISA

3.1 Objeto de estudo

Este trabalho teve como objetivo estudar o desempenho da metaheurística ILS, utilizando a técnica de busca local *Very Large-scale Neighborhood Search* – VLNS (AHUJA *et al*., 2002) para resolver o $Pm||\sum w_i T_i$, de uma nova maneira, ainda não explorada na literatura. Mais especificamente, o trabalho consistiu em implementar a técnica de busca em vizinhança de grande porte VLNS, com cancelamento de ciclos negativos para particionar as tarefas entre as máquinas e combiná-la com a *Dynasearch* para otimizar o sequenciamento das tarefas em cada máquina separadamente. Os dois métodos de busca foram combinados na metaheurística ILS, de acordo com a versão proposta por Della Croce, Garaix e Grosso, (2012).

Para verificar a eficiência desta combinação, também foi implementada a busca VLNS que utiliza o algoritmo de *Matching* para realizar o particionamento das tarefas entre as máquinas, proposta por Della Croce, Garaix e Grosso, (2012). As duas versões foram comparadas em testes computacionais realizados com bases de instâncias cujas soluções ótimas são conhecidas e outras maiores para as quais não são conhecidas as soluções ótimas.

Foram realizados testes computacionais com problemas *benchmark* , disponíveis da literatura, para avaliar a eficiência da implementação proposta. Os resultados confirmaram a

hipótese de que a utilização de buscas em vizinhança de grande porte nas duas etapas do problema podem produzir algoritmos tão eficientes quanto aqueles conhecidos na literatura, passíveis ainda de melhorias nos algoritmos propostos.

3.2 Procedimentos de coleta de dados

As duas versões da ILS foram testadas com dados *benchmark* disponíveis na *internet* e largamente utilizados para se verificar a eficiência de algoritmos que resolvem o problema. Tanaka e Araki, (2008) apresentam soluções ótimas para problemas com 20 e 25 tarefas e a quantidade de máquinas m = 2, 4, ... ,10, cujos dados de entrada e as respectivas soluções podem ser acessadas no sitio https://sites.google.com/site/shunjitanaka/pmtt. Problemas de maiores dimensões podem ser encontrados no sitio http://algox.icomp.ufam.edu.br, sendo que Pessoa *et al.*, (2010) encontraram as soluções ótimas para problemas com m = 2 e 4 máquinas e até 100 tarefas. Por ser um problema *NP-hard*, o tempo de processamento de algoritmos exatos cresce exponencialmente com o tamanho do problema, impossibilitando a obtenção de soluções ótimas para problemas de dimensões maiores.

4. EXPERIMENTOS COMPUTACIONAIS

Os experimentos computacionais foram realizados em um computador pessoal 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 da metaheurística proposta. Posteriormente foram utilizados problemas maiores cujas soluções ótimas não são conhecidas. Os resultados são comparados e analisados ao final.

Cada instância estudada neste experimento foi resolvida 5 vezes e contou com um tempo de execução de 15 minutos, valor escolhido empiricamente.

4.1 Comparação entre o ILS1 e o ILS2 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. Para cada uma das 5 resoluções, foi verificado quantas vezes cada

versão atingiu a solução ótima. Os parâmetros R e T, presentes na Tabela 1, são o *due date range* e o *tardiness fator* respectivamente, que caracterizam e determinam a dificuldade de cada instância (CRAUWELS; POTTS; VAN WASSENHOVE, 1998).

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 e 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	**IL**	**%ot**	0	0	0	0	0,05	0	2,20	0,12	0,27	0,55	0,43	0,11	0	0,50	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

Fonte: Os autores.

Na Tabela 1, pode-se observar que o número de soluções ótimas encontradas pelo ILS1 aumentou quando *m* aumenta 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 de que o ILS1 obtém seus melhores resultados nos problemas em que cada máquina realiza 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.

4.2 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 estudados contam com 50 tarefas (*n* = 50) 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 tempo de processamento de 15 minutos para cada execução. A linha "Min" contém o melhor 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	2,30	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	1,20	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	1,10	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	1,90	1,60	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	

Fonte: Os autores.

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 compromete 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.

4.3 Implicações Práticas do Trabalho

O estudo das duas versões da metaheurística ILS utilizadas para resolver o problema de sequenciamento de tarefas em máquinas paralelas com penalização do atraso, mostra que a versão proposta é tão eficiente quanto a versão conhecida na literatura, sendo que esta última ainda pode ser melhorada com o refinamento da implementação. Na prática, isso significa que ela pode ser utilizada para resolver situações modeladas como um problema clássico do tipo $Pm||\sum w_j T_j$. São situações em que se pretende realizar uma série de tarefas, cada uma com uma data de entrega e que, caso não seja entregue até esta data, incorre em uma multa por dia de atraso. As diferentes tarefas contam com seus respectivos tempos de processamento e podem ser realizadas em um conjunto de máquinas idênticas e que trabalham em paralelo.

Por se tratar de uma metaheurística, a versão proposta é capaz de resolver problemas de qualquer dimensão, e o estudo mostra que para problemas maiores, as soluções obtidas são de boa qualidade. Nos experimentos computacionais, o tempo de processamento foi limitado a quinze (15) minutos, o que é considerado um tempo razoável tanto para o processamento da metaheurística quanto para o usuário. É evidente que para tempos de processamentos maiores, a tendência é que as soluções obtidas sejam melhores, limitadas, é claro, a uma taxa de melhoria decrescente e ao valor da solução ótima.

5. CONCLUSÕES

Neste trabalho foi apresentada uma nova versão da metaheurítica ILS que utiliza técnicas de 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. O diferencial desta versão é que o particionamento das tarefas entre as máquinas é realizado por meio de uma busca do tipo *k-optimal*, onde um vizinho de uma solução é obtido com a realização de uma *troca cíclica* ou de uma *troca em cadeia* das tarefas de uma

série de máquinas, podendo envolver todas as máquinas, se for o caso. Para realizar tal busca de forma eficiente, é construído um grafo de melhoria e neste grafo procura-se por ciclos negativos, os quais levam a uma redução no custo da solução. De acordo com pesquisa realizada pelos autores, esta abordagem não foi testada anteriormente para o problema estudado.

Uma 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 entre as soluções obtidas. Sendo assim, o algoritmo proposto pode ser empregado para resolver problemas de grande porte presentes em uma série de situações práticas, produzindo soluções de alta qualidade.

Por outro lado, este trabalho pode ser continuado, com o aprimoramento da implementação da versão proposta para o ILS, uma vez que ela se mostra bastante promissora e competitiva, e foi testada em sua primeira versão.

Referências

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.

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.

BIGRAS, L. P.; GAMACHE, M.; SAVARD, G. Time-indexed formulations and the total weighted tardiness problem. **INFORMS Journal on Computing**, v. 20, n. 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, n. 3, p. 397-414, 2004.

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

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.

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.

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

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

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 **Handbook of metaheuristics,** p. 320-353, Springer, Boston, MA, 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, n. 3, p. 543-559, 2007.

PESSOA, A.; UCHOA, E.; de ARAGÃO, M. P.; RODRIGUES, R. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. **Mathematical Programming Computation**, v. 2, n. 3-4, p. 259-290, 2010.

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.

TANAKA, S.; ARAKI, M. A branch-and-bound algorithm with Lagrangian relaxation to minimize total tardiness on identical parallel machines. **International Journal of Production Economics**, v. 113, n. 1, p. 446-458, 2008.