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

Eduardo de Oliveira Ferreira
Universidade Federal de Ouro Preto – Departamento de Eng. De Controle e Automação
Campus Morro do Cruzeiro, Ouro Preto, MG – Brasil - 35400-000
deoliveira.eng@gmail.com

Gustavo Peixoto Silva
Universidade Federal de Ouro Preto – Departamento de Computação
Campus Morro do Cruzeiro, Ouro Preto, MG – Brasil - 35400-000
gustavo@iceb.ufop.br

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. Existem duas questões na resolução do problema: o particionamento das tarefas entre as máquinas e o sequenciamento das tarefas nas máquinas. A contribuição deste trabalho é resolver as duas etapas com heurísticas de busca de grande porte. A técnica *Very Large-scale Neighborhood Search*, que usa grafos de melhoria, é empregada para realizar o particionamento das tarefas, e um algoritmo de Programação Dinâmica *Dynasearch* realiza o sequenciamento das tarefas nas máquinas. Ambas as buscas são combinadas na metaheurística ILS e os resultados são comparados com aqueles obtidos utilizando o ILS-VLNS combinado com o método de busca local *First Improvement.*

PALAVARAS CHAVE: Máquinas paralelas. Busca de grande porte. *Dynasearch*
Área principal: Metaheuristicas.

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 so that each task is performed by a single machine, each machine executes a single task at a time while minimizes the total weighted tardiness. There are two fundamental questions to be handled: the partitioning of tasks between machines and the tasks 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 employes improvement graphs, is adopted to perform the tasks partiotioning and a Dynamic Programming algorithm, called Dynasearch, performs task sequencing on each machines. Both search heuristics are combined in an ILS metaheuristic and the results are compared with the results obtained with ILS-VLNS that uses First Improvement Search on each single machine.

KEYWORDS. Parallel machines. Very large-scale search. Dynasearch.

Main area: Metaheuristics.

1. Introdução

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 vem sendo propostas (Ahuja *et al.* 2002). O sequenciamento de tarefas é um problema comum, as tarefas devem ser inicialmente particionadas entre as máquinas e posteriormente sequenciadas dentro em cada uma delas. 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* , denotado por $Pm||\sum w_j T_j$ (Della Croce *et al*. 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 *et al.* 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. O problema aqui estudado é, segundo (Lenstra *et al*. 1977) e (Du e Leung, 1990), do tipo *NP-Hard*. Isso significa que não se conhece algoritmo de tempo polinomial que consiga encontrar a solução ótima.

Um dos problemas enfrentados na implementação de algoritmos 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 (Ahuja *et al.* 2002) integrada com a metaheuristica *Iterated Local Search* – ILS (Lourenço *et al*. 2003), como proposto por (Della Croce *et al.* 2012). Esse procedimento já foi testado anteriormente por (Franco e Silva 2016), utilizando o método de busca local clássico *First Improvement* para o sequenciamento das tarefas em cada máquina. 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 denominado *Dynasearch*, por (Congran *et al.* 2002). 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 *et al*. 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* pode realizar várias trocas por iteração, o que impede que a busca seja atraído para ótimos locais pobres e faz com que ela consiga encontrar ótimos melhores do que os métodos tradicionais quando executado separadamente em cada máquina (Congram *et al.* 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, e os resultados foram comparados o ótimo global, quando conhecido, e com aqueles obtidos por (Franco e Silva 2016).

Este trabalho está dividido da seguinte forma: na Secção 2 são apresentados os seus objetivos, 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. Objetivos do Trabalho

Este trabalho teve como objetivo estudar e implementar uma metaheurística baseado na técnica de busca local *Very Large-scale Neighborhood Search* – VLNS (Ahuja *et al*., 2002) para resolver o $Pm||\sum w_i T_i$ para o qual existem relativamente pouco trabalhos desenvolvidos (Della Croce *et al.* 2012). O objetivo específico foi implementar a técnica de busca em vizinhança de grande porte *Dynasearch* para otimizar o sequenciamento das tarefas em cada máquina

separadamente. Esta técnica é uma variante do VLNS e foi proposta por (Congram *et al*. 2002) e cuja complexidade foi melhorada por (Ergun e Orlin 2006). Nos testes computacionais, (Congram *et al*. 2002) obtiveram resultados superiores àqueles encontrados por técnicas clássicas de busca local.

A primeira etapa do problema, ou seja, o particionamento das tarefas entre as máquinas foi realizado pelo VLNS que constrói um *grafo de melhoria* e busca por ciclos negativos neste grafo. A 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*.

Para otimizar o sequenciamento das tarefas em cada máquina foi aplicada a busca de grande porte *Dynasearch*, implementada por meio de um algoritmo de Programação Dinâmica. Finalmente, as duas etapas do problema foram combinadas em uma versão da metaheurística *Iterated Local Search* – ILS. 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 podem melhorar os resultados obtidos.

3. Método de Resolução do Problema

Para problemas do tipo $Pm||\sum w_j T_j$ é dado um conjunto de tarefas $N = \{1, 2, ..., n\}$, assim como o tempo de processamento p_j, pesos w_j além das datas de entrega d_j especificada para cada tarefa $j \in N$. 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 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 *et al*. 2008), (Pan e Shi 2007), (Rodrigues *et al*. 2008), (Congram *et al*. 2002) e (Grosso *et al*. 2004). A literatura parece ser razoavelmente limitada para o problema de máquinas paralelas. Os trabalhos mais recentes 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).

3.1 Particionamento das tarefas entre as máquinas via *Very Large Local Search - VLNS*

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 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 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 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 $O(n^2m)$.

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 é uma solução ótima local segundo esta vizinhança. Esta heurística de busca local foi implementada e testada por Rodrigo e Silva (2016) que traz mais detalhes para o leitor interessado.

3.2 Vizinhança *Dynasearch Swap* em uma 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. 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 que 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 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 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 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 $\sigma(1)$, ..., $\sigma(k)$ 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 sequencia 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.

3.3 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 $\sigma(1)$, ..., $\sigma(k)$ 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:

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

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.

3.4 O Algoritmo utilizando a vizinhança *Dynasearch Swap*

A implementação do algoritmo com busca na vizinhança *Dynasearch Swap* 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 Swap*. Usando o algoritmo de Programação Dinâmica apresentado na seção anterior, 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 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 $F(\sigma_n^{t-1}) \leq F(\sigma_n^{t-2})$ então deve ser executada uma nova iteração tendo σ^t como a solução corrente.

3.5 Geração da Solução Inicial

A solução inicial, utilizada neste trabalho, foi obtida empregando a estratégia do *Early 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 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 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, e conta com as duas heurísticas de busca local: a busca em N_1, realizada pela função *descida1*(), na linha 4 do Algoritmo 1, tenta melhorar o sequenciamento das tarefas de cada máquina isoladamente pela vizinhança *Dynasearch Swap*. A busca em N_2, feita pela função *descida2*() 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 considerando movimentos entre máquinas. Nas linhas 19 e 22 ocorre a perturbação intra-máquinas aplicando a função *kick1*( ), ou seja, são modificadas as posições das tarefas em máquinas escolhidas aleatoriamente. Posteriormente, na linha 22, é realizada a perturbação via *kick2*( ), que consiste em aplicar 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 = descida1(S, N1);
 5.       Repita
 6.           S2 = descida2(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*;
```

Algoritmo 1. Pseudocódigo da implementação do ILS.

4. Apresentação e Análise dos Resultados

Para gerar a solução inicial, foi utilizada a já conhecida heurística gulosa *Earliest Due Date*, considerando as data 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 da metaheurística proposta. Posteriormente foram utilizados problemas maiores cujas soluções ótimas não são conhecidas, mas foram comparadas com aquelas obtidas pela versão ILS-VLNS.

4.1 Testes de Eficiência da Metaheurística ILS - *Dynasearch Swap + VLNS*

Após a validação da busca *Dynasearch Swap*, a heurística 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. Utilizando a metaheurística ILS-*Dynasearch* + VLNS, denominada doravante apenas ILS-DVLNS, 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. Para cada problema, a metaheurística ILS foi executada 5 vezes, com tempo máximo de 15 minutos para cada execução, e os resultados foram comparados com aqueles que foram obtidos usando o 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*.

Conforme pode ser observado na Tabela 1, para 4 máquinas, o ILS-DVLNS encontrou a melhor média dentre para 7 problemas, contra 2 problemas cujas melhores médias foram encontradas pelo ILS-VLNS, além de 1 empate. Os melhores resultados médios são destacados em negrito.

Tabela 1. Resultados obtidos utilizando ILS-VLNS e ILS-DVLNS para problemas com $n = 50$ e $m = 4$, sem soluções ótimas conhecidas

	ILS-DVLNS				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 2, o ILS-DVLNS 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 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 teve uma ligeira redução na sua eficiência.

Tabela 2. Resultados obtidos utilizando ILS-VLNS e ILS-DVLNS para problemas com $n = 50$ e $m = 10$, sem soluções ótimas conhecidas

	ILS-DVLNS				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 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 3. 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, ele foi executado para outros problemas cujas soluções ótimas são conhecidas.

Tabela 3. Resultados obtidos utilizando ILS-VLNS e ILS-DVLNS para problemas com $n = 20$ e $m = 4$, com soluções ótimas conhecidas

Problemas			ILS – VLNS				ILS-DVLNS			
#	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

Todos os problemas dos testes seguintes contam com 20 tarefas, sendo que os problemas da Tabela 4 contam com 2 máquinas, os problemas da Tabela 5 contam 6 máquinas e finalmente, na Tabela 6 são apresentados os resultados dos problemas que utilizam 8 máquinas para sequenciar as 20 tarefas. Para todos os problemas, o ILS-DVLNS 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 conseguiu chegar à solução ótima na maioria dos problemas com 2 máquinas (Tabela 4) e em todos os problemas com 6 máquinas (Tabela 5). Contudo, alcançou o ótimo em apenas 6 dos 15 problemas com 8 máquinas (Tabela 6). Este resultado pode ser um indício de que uma quantidade muito baixa de tarefas por máquinas pode comprometer o desempenho das ILS-DVLNS.

Tabela 4. Resultados obtidos utilizando o ILS-DVLNS para problemas com $n = 20$ e $m = 2$, cujas soluções ótimas são conhecidas

Problemas			ILS-DVLNS			
#	R	T	%otm	Média	DP	QT
1	0,2	0,2	0	147	0	5
2	0,2	0,6	0	0	0	5
3	0,2	1,0	0	0	0	5
4	0,4	0,2	0	695	0	5
5	0,4	0,6	0	341	0	5
6	0,4	1,0	0	227	0	5
7	0,6	0,2	0	1741	0	5
8	0,6	0,6	**0,45**	1557	0	0
9	0,6	1,0	0	1290	0	5
10	0,8	0,2	0	3186	0	5
11	0,8	0,6	0	2768	0	5
12	0,8	1,0	**0,15**	1997	0	0
13	1,0	0,2	**0,04**	4653	0	0
14	1,0	0,6	0	3667	0	5
15	1,0	1,0	0	2768	0	5

Tabela 5. Resultados obtidos utilizando o ILS-DVLNS para problemas com $n = 20$ e $m = 6$, com soluções ótimas conhecidas

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

Utilizando os dados das Tabelas 3, 4, 5 e 6, foi montado o gráfico da Figura 1 que tem no eixo das abscissas o número de máquinas e nas ordenadas o número de vezes que o ILS-DVLNS encontrou a solução ótima. Pode-se observar, através dos pontos, que o número de soluções ótimas encontradas pelo ILS-DVLNS subiu 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. Observando a *linha de tendência* que foi gerada utilizando os quatro pontos do gráfico tem-se uma reta com inclinação negativa, indicando que o aumento no número maior de máquinas pode acarretar em uma diminuição no número de soluções ótimas encontradas. Desta forma, há evidências que o ILS-DVLNS é mais eficiente para problemas nos quais cada máquina apresenta um grande número de tarefas, ou seja, a razão entre o total de tarefas dividido pelo total de máquinas é grande. Resta destacar que, embora o ILS-DVLNS 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.

Tabela 6. Resultados obtidos utilizando o ILS-DVLNS para problemas com $n = 20$ e $m = 8$, com soluções ótimas conhecidas

Problemas			ILS-DVLNS			
#	R	T	%otm	Média	DP	QT
1	0,2	0,2	0	177	0	5
2	0,2	0,6	0	77	0	5
3	0,2	1,0	0	100	0	5
4	0,4	0,2	0	439	0	5
5	0,4	0,6	**0,05**	422,2	0,45	4
6	0,4	1,0	0	431	0	5
7	0,6	0,2	**2,2**	790	0	0
8	0,6	0,6	**0,12**	823	0	0
9	0,6	1,0	**0,27**	734	0	0
10	0,8	0,2	**0,55**	1282	0	0
11	0,8	0,6	**0,43**	1168	0	0
12	0,8	1,0	**0,105**	953	0	0
13	1,0	0,2	0	1678	0	5
14	1,0	0,6	**0,5**	1420	0	0
15	1,0	1,0	**0,43**	1168	0	0

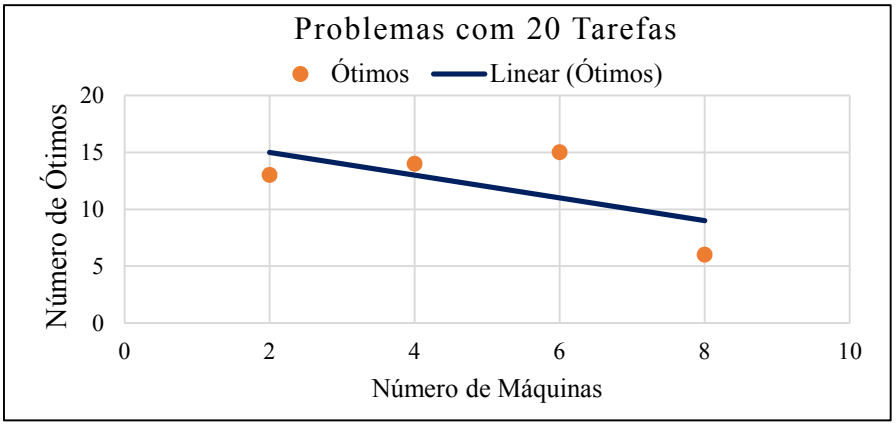

Figura 1. Gráfico com linha de tendência do número de ótimos encontrados pelo ILS-DVLNS para 20 tarefas em 2, 4, 6 e 8 máquinas

5. Conclusões

Neste trabalho foram empregadas heurísticas de busca em vizinhança de grande porte nas duas etapas do problema. O VLNS foi utilizado para realizar a alocação das tarefas às máquinas e o *Dynasearch Swap* foi usado para definir a melhor sequência das tarefas em cada máquina. As duas heurísticas de busca local foram combinadas em uma versão da metaheurística ILS, denominada ILS-DVLNS. Esta nova versão foi comparada com a versão ILS-VLNS, que usa busca em vizinhança de grande porte apenas na primeira etapa, empregando a busca clássica do tipo *First Improvement* na segunda etapa.

Utilizando a metaheurística ILS-DVLNS 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 um ligeira redução. Esta diferença pode indicar que o ILS-DVLNS é mais efetivo quando o número de tarefas por máquinas é maior. Por outro lado, ao analisar problemas com 20 tarefas e 4 máquinas, com soluções ótimas conhecidas, não se obteve diferença significativa entre o as duas metaheuristicas, com as duas atingindo a solução ótima da maioria dos problemas. Apesar do objetivo das metaheuristicas 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 que a outra.

Ao investigar a eficiência do ILS-DVLNS 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 conseguiu encontrar a solução ótima em todas as execuções. Entretanto, ao aumentar o número de máquinas para 8, 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. Não obstante, o método foi capaz de obter soluções muito próximas do valor ótimo, com um gap médio de 0,52% para este caso, donde pode-se concluir que a versão apresentada neste trabalho é bastante competitiva produzindo resultados muito próximos das soluções ótimas.

Agradecimentos

Os autores agradecem à FAPEMIG e à UFOP pelo apoio recebido no desenvolvimento deste trabalho.

Referências Bibliográficas

Ahuja, R. K., Ergun, Ö., Orlin, J. B. e Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, *123*(1-3), 75–102.

Anghinolfi, D. e Paolucci, M. (2007). Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. *Computers & Operations Research*, *34*(11), 3471-3490.

Arenales, M., Armentano, V., Morabito, R. e Yanasse, H. (2011). Pesquisa Operacional. Elsevier Brasil.

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

Bigras, L. P., Gamache, M. e Savard, G. (2008). Time-indexed formulations and the total weighted tardiness problem. *INFORMS Journal on Computing*, *20*(1), 133-142.

Bilge, Ü., Kıraç, F., Kurtulan, M. e Pekgün, P. (2004). A tabu search algorithm for parallel machine total tardiness problem. *Computers & Operations Research*, *31*(3), 397-414.

Blazewicz, J., Drabowski, M. e Weglarz, J. (1986). Scheduling Multiprocessor Tasks to Minimize Schedule Length. *IEEE Transactions on Computers*, (35), 389-393.

Congram, R. K., Potts, C. N. e van de Velde, S. L. (2002). An iterated *Dynasearch* algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, *14*(1), 52-67.

Della Croce, F., Garaix, T. e Grosso, A. (2012). Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem. *Computers & Operations Research*, 39(6), 1213–1217.

Du, J. e Leung, J. Y. T. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3), 483-495.

Ergun, Ö. e Orlin, J. B. (2006). Fast neighborhood search for the single machine total weighted tardiness problem. *Operations Research Letters*,*34*(1), 41-45.

Franco, R. R. e Silva, G. P. (2016) A metaheuristic Iterated Local Search and Very Large-scale Neighborhood Search to solve the Parallel Machine Scheduling Problem. *In: Proceedings of the XVIII Latin-Iberoamerican Conference on Operations Research*, 1, 327-334.

Graham, R. L., Lawler, E. L., Lenstra, J. K. e Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, *5*, 287-326.

Grosso, A., Della Croce, F. e Tadei, R. (2004). An enhanced *Dynasearch* neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, *32*(1), 68-72.

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

Lenstra, J. K., Rinnooy Kan, A. H. G. e Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1, 343–362.

Lourenço, H. R., Martin, O. C. e Stützle, T. (2003). Iterated local search. In *Handbook of metaheuristics,* (320-353). Springer US.

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

Rodrigues, R., Pessoa, A., Uchoa, E. e Poggi de Aragão, M. (2008). Heuristic algorithm for the parallel machine total weighted tardiness scheduling problem. *Relatório de Pesquisa em Engenharia de Produção*, 8(10).