Método *Proximity Search* para a resolução do problema de flow shop scheduling não permutacional com trabalhadores heterogêneos (FSNPTH).

Matheus de Freitas Araujo
Universidade Federal de Viçosa
Campus Universitário, 36570-900, Viçosa-MG, Brasil
matheus.f.araujo@ufv.br

José Elias Claudio Arroyo
Universidade Federal de Viçosa
Campus Universitário, 36570-900, Viçosa-MG, Brasil
jarroyo@dpi.ufv.br

André Gustavo dos Santos
Universidade Federal de Viçosa
Campus Universitário, 36570-900, Viçosa-MG, Brasil
andre@dpi.ufv.br

RESUMO

Esse trabalho aplicar o método *Proximity Search* (PS) para a resolução do problema FSNPTH. Este problema pode ser classificado como um problema multicomponente, uma vez que trata de dois problemas distintos na qual a resolução de um implica diretamente no resultado do outro. O problema consiste em: alocar trabalhadores heterogêneos em máquinas, nas quais a heterogeneidade dos trabalhadores é definida em relação ao tempo gasto para operar uma determinada máquina; sequenciar tarefas em máquinas em séries de forma que minimize o tempo de conclusão da última tarefa no processo (makespan). PS é um método genérico que faz uso do modelo de Programação Inteira do problema para melhorar iterativamente a solução. O PS modifica a função objetivo para reduzir o espaço de busca. Neste trabalho, o desempenho do PS é comparado com o solver CPLEX. Os resultados mostraram que o PS determina boas soluções em curto espaço de tempo.

PALAVRAS CHAVE. Flow Shop Scheduling, *Proximity Search*, Programação Matemática.

IND - PO na Indústria, OC - Otimização Combinatória, PM - Programação Matemática

ABSTRACT

This work aims to apply the method Proximity Search (PS) to solve the problem FSNPTH. This problem can be classified as a multicomponent problem, since these are two separate problems, which the resolution of a directly gives the results of another. The problem consists in: allocating heterogeneous workers in machines in which the heterogeneity of workers is defined in relation to the time spent to operate a particular machine; sequencing jobs on machines in series that minimizes the time to complete the last task in the process (makespan). PS is a generic method that makes use of the problem of Integer Programming model to iteratively improve a reference solution. The PS changes the objective function to reduce the search space. In this study, the performance of the PS is compared with the CPLEX solver solving the original problem model. The results showed that the PS provides good solutions in a short time.

KEYWORDS. Flow Shop Scheduling, Proximity Search, Mathematical Programming.

IND - PO Industry, CO - Combinatorial Optimization, PM - Mathematical Programming

1. Introdução

Os problemas de flow shop scheduling ganharam espaço não só na academia, mas também no mercado de trabalho, uma vez que esses problemas refletem diretamente no sistema de produção de diversas empresas. Entretanto, apenas os problemas de flow shop scheduling não têm atendido completamente as empresas, uma vez que este problema busca apenas alocar de maneira eficiente um conjunto de tarefas não preemptivas em um conjunto de máquinas dispostas em série, visando minimizar algumas medidas de desempenho, tal como o tempo máximo de conclusão (Makespan). Sendo assim tem-se tornado cada vez mais comum a combinação de dois ou mais problemas, para suprir melhor a necessidade de resolução dos problemas enfrentados no cotidiano das empresas. Um exemplo disso é a abordagem apresentada por [Benavides et al. 2014] que combina o problema de flow shop não permutacional com a designação de trabalhadores com diferentes habilidades em postos de trabalhos ou máquinas.

O problema de flow shop não permutacional com trabalhadores heterogêneos (FSNPTH) surgiu a partir da necessidade de alocação de trabalhadores deficientes com diferentes habilidades em postos de trabalhos, de forma que a heterogeneidade dos trabalhadores interfira o mínimo possível no resultado final do sistema de produção.

Segundo o Instituto Brasileiro de Geografia e Estatística – IBGE, 23% da população brasileira sofre de alguma deficiência (visual, auditiva, motora, mental ou intelectual) [Brasil 2010]. O IBGE ainda afirma que existem cerca de 23,7 milhões pessoas com deficiência em idade ativa desempregadas, mesmo com as políticas para integração de pessoas portadoras de deficiência, como a lei 7.853 que assegura o acesso ao trabalho. Esses números comprovam a relevância desse trabalho, visto que os deficientes precisam ser eficientes e competitivos ao exercer suas atividades, não só pela sua sobrevivência no mundo, mas também para serem capazes de crescer no mercado de trabalho. Com isso visa-se melhorar a inclusão social dos deficientes, designando-os em postos de trabalhos nos quais suas deficiências interfiram o minimo na atividade.

Na literatura, diversos pesquisadores abordam o problema de flow shop scheduling tais como: Ruiz e Maroto [Ruiz e Maroto 2005] e Potts e Trusevich [Potts e Strusevich 2009]. Muitos autores apresentam metaheuristicas para resolução do problema de maneira aproximada, como: Simulated Annealing [Tandon et al. 1991], Iterated Greedy [Ying 2008], Busca Tabu [Brucker et al. 2003, Liao et al. 2006], Colônia de Formigas [Yagmahan e Yenisey 2010, Rossi e Lanzetta 2013] e algoritmos híbridos que combinam diferentes meta-heuristicas [Lin e Ying 2009].

Esse trabalho apresenta e comprova a eficiência do método *Proximity Search* proposto por [Fischetti e Monaci 2014] para a resolução do problema FSNPTH.

2. Definição do Problema

O problema FSNPTH definido por [Benavides et al. 2014] consiste em alocar trabalhadores heterogêneos à máquinas, nas quais a heterogeneidade dos trabalhadores é definida em relação ao tempo que cada um gasta para operar uma determinada máquina, e a sequenciar tarefas nas máquinas em séries de forma que o makespan seja minimizado.

Sendo assim, existe um conjunto de trabalhadores $W = \{1, 2, ..., w\}$ que deve ser alocado em um conjunto $M = \{1, 2, ..., m\}$ de máquinas, sendo $w = m$, ou seja, cada trabalhador deve ser alocado em uma única máquina. Definida a alocação de trabalhadores às máquinas, tem-se o problema de flow shop scheduling não permutacional, onde existe um conjunto $N = \{1, 2, ..., n\}$ de tarefas na qual que devem ser processadas no conjunto máquinas dispostas em séries. Cada tarefa j é composto por um conjunto de m operações consecutivas $O_{1j}, O_{2j}, ..., O_{mj}$, onde O_{ij} é a operação da tarefa j na máquina i. O tempo de processamento de cada operação das tarefas depende diretamente do trabalhador alocado à máquina, ou seja, se o trabalhador k é alocado à máquina i, o tempo da operação O_{ij} é t_{ijk}. Com isso busca-se encontrar um conjunto π com m sequências (uma para cada máquina) que minimize o makespan. O problema FSNPTH é NP-Difícil, já que o problema de flow shop scheduling permutacional com $m \geq 3$ é NP-Difícil [Garey et al. 1976]. Na Tabela 1 é apresentada uma instância para o problema FSNPTH com 4 tarefas, 4 máquinas e 4

trabalhadores, no qual os valores representam o tempo de processamento de cada tarefa j em cada máquina i dado um trabalhador k (t_{ijk}). Os valores de $t_{ijk} = inf$ indicam que o trabalhador k não pode ser alocado (ou operar) à máquina i.

Tabela 1: Instância do problema de flow shop scheduling não permutacional com trabalhadores heterogêneos.

Tarefas	m_1				m_2				m_3				m_4			
	w_1	w_2	w_3	w_4	w_1	w_2	w_3	w_4	w_1	w_2	w_3	w_4	w_1	w_2	w_3	w_4
j_1	1	2	3	1	2	4	2	inf	2	4	2	2	2	2	2	2
j_2	1	1	4	3	1	1	4	inf	2	2	2	4	2	1	3	4
j_3	2	1	2	3	2	2	1	inf	2	1	2	4	4	2	2	3
j_4	2	5	2	3	1	4	2	inf	1	4	2	2	2	3	2	1

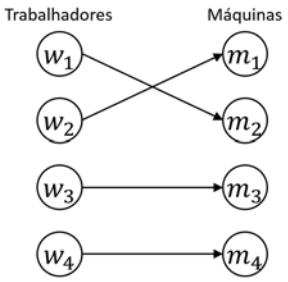


Figura 1: Alocação dos trabalhadores para a instância acima.

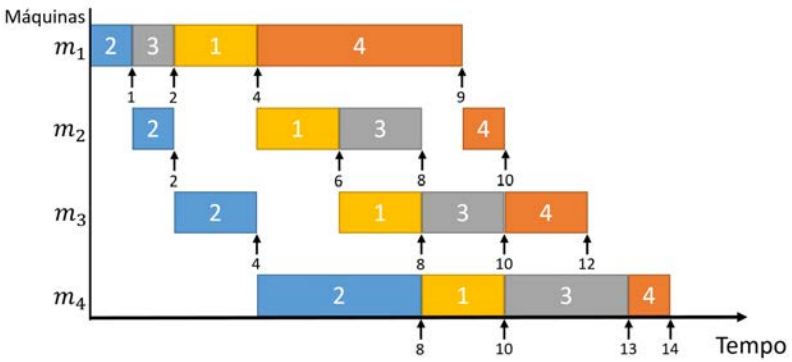


Figura 2: Alocação ótima das tarefas para a instância acima.

Na Figura 1 é apresentada a designação dos trabalhadores às máquinas e na Figura 2 é apresentado o sequenciamento ótimo das tarefas nas máquinas da solução ótima para a instância em questão. A Figura 2 apresenta o conjunto π de m sequências do problema FSNPTH com o valor de makespan igual a 14 ($C_{max} = 14$). A sequência $\pi_1 = \{2, 3, 1, 4\}$ é executada na máquina 1, $\pi_2 = \{2, 1, 3, 4\}$ é executada na máquina 2, as sequências $\pi_3 = \{2, 1, 3, 4\}$ e $\pi_4 = \{2, 1, 3, 4\}$ são executadas nas máquinas 3 e 4 respectivamente. Observa-se que mesmo a designação não sendo ótima, levando em consideração a média ou a soma dos tempos de processamentos das tarefas, o valor makespan (tempo de conclusão da última tarefa no processo) é mínimo para este sequenciamento. Isso acontece por se tratar de um problema multicomponente, ou seja, um problema que consiste em dois ou mais problemas individuais [Bonyadi et al. 2013]. Em problemas dessa natureza duas características podem ser observadas: a combinação e a interdependência. A combinação

é a característica que determina que um problema seja a combinação de mais de um problema, já a interdependência faz com que os problemas interfiram no resultado uns dos outros.

3. Modelagem Matemática de Programação Inteira Mista

Em [Benavides et al. 2014] é proposta uma modelagem matemática para o problema FSNPTH na qual existem m máquinas, w trabalhadores, (onde $m = w$) e n tarefas. O modelo apresentado a seguir utiliza o seguinte parâmetro:

- MG é um valor suficientemente grande definido por $\sum_{i=1}^{m}\sum_{j=1}^{n} max(t_{ijk} \forall k \in W)$;

O modelo utiliza as seguintes variáveis de decisão:

- C_{max}: Tempo máximo de conclusão das tarefas ou makespan;
- x_{ij}: Tempo de início da tarefa j na máquina i;
- p_{ij}: Tempo de processamento da tarefa j na máquina i;
- z_{ik}: Define a alocação do trabalhador k na máquina i, sendo $z_{ik} = 1$ se o trabalhador k é alocado a máquina i e $z_{ik} = 0$ caso contrário;
- $y_{ijj'}$: Define a precedência das tarefas, sendo $y_{ijj'} = 1$ se a tarefa j precede a tarefa j' na máquina i e $y_{ijj'} = 0$ caso contrário;

O modelo é apresentado a seguir:

$$\min C_{max} \tag{1}$$

sujeito a:

$$x_{Mj} + p_{Mj} \leq C_{max}, \qquad \forall j \in N \tag{2}$$
$$x_{ij} + p_{ij} \leq x_{i+1j}, \qquad \forall i \in M-1, \forall j \in N \tag{3}$$
$$x_{ij} + p_{ij} \leq x_{ij'} + MG(1 - y_{ijj'}), \qquad \forall i \in M, \forall j \neq j' \in N \tag{4}$$
$$y_{ijj'} + y_{ij'j} = 1, \qquad \forall i \in M, \forall j \neq j' \in N \tag{5}$$
$$p_{ij} = \sum_{k=1}^{w} t_{ijk} z_{ik} \qquad \forall i \in M, \forall j \in N \tag{6}$$
$$\sum_{k=1}^{w} z_{ik} = 1, \qquad \forall i \in M \tag{7}$$
$$\sum_{i=1}^{m} z_{iw} = 1, \qquad \forall w \in W \tag{8}$$
$$x_{ij} \geq 0, \qquad \forall i \in M, \forall j \in N \tag{9}$$
$$y_{ijj'} \in \{0,1\}, \qquad \forall i \in M, \forall j \neq j' \in N \tag{10}$$
$$z_{ik} \in \{0,1\}, \qquad \forall i \in M, \forall k \in W \tag{11}$$

A função objetivo denifida em (1) é minimizar o valor do makespan (C_{max}). A restrição (2) determina o C_{max}. O conjunto de restrições (3) e (4) dizem respeito ao tempo de início de cada tarefa em cada máquina de acordo com a precedência de tarefa. A restrição de precedência das tarefas é definida pela Equação (5). A restrição (6) define o valor de p_{ij} a partir da alocação dos trabalhadores. O conjunto de restrições (7) e (8) garante a designação dos trabalhadores aos postos de trabalhos ou máquinas. A restrição (9) define a não negatividade das variáveis x_{ij}. O conjunto de restrições (10) e (11) define $y_{ijj'}$ e z_{ik} como variáveis binárias.

4. Proximity Search

O *Proximity Search* (PS) é um método proposto por [Fischetti e Monaci 2014] para resolver problemas de programação linear inteira mista 0-1 (PLIM). Este método utiliza uma estratégia de resolução cujo objetivo é, a partir de uma solução inicial viável, encontrar uma sequência de soluções melhoradas para um dado modelo de PLIM do problema. O funcionamento do PS consiste em substituir a função objetivo do modelo de programação inteira mista por uma função de proximidade e utilizar a resolução deste novo modelo como heurística de refinamento. A proposta do PS é suprir a deficiência dos métodos enumerativos, geralmente utilizados para resolução de problemas de PLIM, para quais não consegue encontrar soluções de boa qualidade em um curto espaço de tempo. Sendo assim, o PS tem se mostrado mais eficiente para resolver problemas de grande porte do que os métodos enumerativos como o branch-and-bound.

O PS inicia a busca a partir de uma solução inicial x', geralmente encontrada por uma heurística, após é adicionada uma restrição de corte (definida pela Equação 12) ao modelo de PLIM. Sendo x e x' duas soluções para o problema, $f(x)$ e $f(x')$ representam os valores do makespan (funções objetivos) e $\theta \geq 0$ é uma constante denominada parâmetro de corte.

$$f(x) \quad \geq \quad f(x') - \theta, \tag{12}$$

Essa restrição garante que apenas soluções melhores que a solução corrente possam ser encontradas quando o modelo é resolvido, com isso o espaço de busca é reduzido consideravelmente.

O próximo passo do PS é substituir a função objetivo do modelo por uma função de proximidade. Os autores [Fischetti e Monaci 2014] propõem a utilização da distância de Hamming como função de proximidade. A ideia por trás da função de proximidade é guiar o algoritmo pelo espaço busca. Com isso aumenta-se a chance de encontrar soluções melhores que a solução inicial. A distância de Hamming é definida pela Equação (13) e busca determinar a distância entre as soluções x e x' formadas por um conjunto de variáveis binárias. Nesse caso, as variáveis binárias $y_{ijj'}$ e z_{ik}, definidas no modelo apresentado na seção 3 são utilizadas para definir a distância entre as soluções x e x'. A distância de Hamming na prática determina o número de variáveis binárias cujo valores sejam distintos entre as soluções x e x'. Em um problema com 2 trabalhadores e 2 máquinas por exemplo, se a solução x tem o trabalhor 1 alocado na máquina 2 ($z_{21} = 1$ e $z_{11} = 0$) e na solução x' o trabalhador 1 é alocado na máquina 1 ($z_{11} = 1$ e $z_{21} = 0$), então o valor de $\Delta(x, x')$ é incrementado em 2 uma vez que as variáveis z_{11} e z_{21} possuem valores distintos nas soluções x e x'. Se as soluções x e x' forem iguais $\Delta(x, x') = 0$.

$$\Delta(x, x') \quad = \quad \sum_{j \in J : x'_j = 0} x_j \quad + \sum_{j \in J : x'_j = 1} (1 - x_j) \tag{13}$$

Algorithm 1 Proximity Search.

1: $x' \leftarrow$ Solução Inicial
2: **enquanto** critério de parada não seja atingido **faça**
3: Adiciona a restrição de corte $f(x) \geq f(x') - \theta$ ao modelo
4: Substituir a função objetivo $f(x)$ do modelo pela função de proximidade $\Delta(x, x')$
5: $x^* \leftarrow$ executa o solver CPLEX até que a condição de parada seja atingida.
6: **se** x^* é uma solução viavel **então**
7: $x^* \leftarrow argmin\{f(x) : g(x) \geq 0, x_j = x^*_j, \forall j \in J\}$
8: **fim se**
9: Atualizar $\Delta(x, x')$ fazendo $x' \leftarrow x^*$ e/ou atualiza θ.
10: **fim enquanto**

O pseudocódigo do *Proximity Search* é descrito pelo Algoritmo 1. O PS necessita de uma solução inicial (Linha 1), então na Linha 3 é adicionada a restrição de corte baseada no valor da solução corrente (x') e o valor de θ. Na Linha 4 a função objetivo do modelo de programação inteira mista é substituída pela função de proximidade $\Delta(x, x')$, que é definida de acordo com a distância de Hamming. Em seguida, na Linha 5 o solver IBM ILOG CPLEX é utilizado para resolver o novo modelo. Na Linha 6 é verificado se o solver Cplex encontrou alguma solução para o novo modelo, caso encontre, na Linha 7 é calculado o valor da função objetivo $f(x^*)$ do modelo original. Em seguida é necessário atualizar a função de proximidade $\Delta(x, x')$ bem como a solução corrente, que deve ser substituída pela solução encontrada pelo modelo (Linha 9).

A partir do algoritmo descrito anteriormente, foram implementadas três versões do PS, PS_1, PS_2 e PS_{2RINS} [Fischetti e Monaci 2014]. O PS_1 é o Algoritmo 1, entretanto nas versões PS_2 e PS_{2RINS} a função de corte apresentada pela Equação (12) é substituída pela função de corte definida na Equação (14), onde z é uma variável de folga contínua ($z >= 0$) e a função de proximidade é substituída pela função de proximidade definida na Equação (15) onde M é um valor positivo suficientemente grande se comparado com valor obtido por Δ (nesse caso é usado $M = (j^2 * m + m^2 + 1)$, que é o número de variáveis binárias utilizadas no modelo mais um). Com a variável de folga z aplicada na função de proximidade e na restrição de corte, o valor da solução incumbente do modelo é rapidamente definida e pode ser igual à solução corrente x'. Essa situação seria inválida, pois espera-se encontrar uma solução x melhor e diferente de x'. Para isso, a constante M vai penalizar a solução com uma função de proximidade muito alta. Ter uma solução incumbente logo no início da resolução do modelo matemático pode melhorar o desempenho e permitir a utilização de heurísticas de refinamento implementadas internamente pelo CPLEX.

$$f(x) \quad \geq \quad f(x') - \theta + z \tag{14}$$

$$\Delta(x, x') + Mz \tag{15}$$

A versão PS_{2RINS} é diferente das outras duas versões, uma vez que utiliza a heurística de refinamento RINS(Relaxation Induced Neighborhood Search), introduzida por [Danna et al. 2005]. A biblioteca ILOG Concert (utilizada para nas implementações do PS) por padrão já implementa essa heurística, sendo assim, nessa versão do PS, o RINS é apenas habilitada durante a execução.

5. Experimentos computacionais

Os experimentos computacionais foram realizados a fim de demonstrar o desempenho do PS em relação a resolução do modelo sem essa técnica. Nos testes foram executadas as 3 versões do PS (PS_1, PS_2 e PS_{2RINS}) e o MODELO. O MODELO refere-se à implementação e resolução do modelo matemático de programação inteira mista definido anteriormente. Esse método então é utilizado posteriormente para realizar as comparações. As implementações do MODELO e do PS foram desenvolvidas utilizando a biblioteca ILOG Concert, que permite implementar modelos matemáticos na linguagem C++ e então resolver pelo solver IBM ILOG CPLEX.

Os experimentos computacionais foram realizados em um Cluster onde cada nó contém 2 processadores Intel(R) Xeon(R) CPU X5650 (12M Cache, 2.66 GHz, 6.40 GT/s Intel(R) QPI, 6 cores, 12 threads) e 24 GB de RAM DDR3 1333 MHz. Entretanto os experimentos foram executados utilizando apenas 1 processador, 4 threads e 8GB de memória RAM. Em todos os experimentos foram fixados o tempo de execução em 3600 segundos para cada instância e o valor de θ foi fixado em 1 (garantindo que o PS prove a otimalidade da solução).

Para cada uma das versões do PS, foram executados testes variando os pesos das variáveis que representam os trabalhadores (z_{ik}) na função de proximidade. Foram testados 4 pesos (1, 10, 50, 100). Por se tratar de um problema multicomponente, ao atribuir pesos nas variáveis dos trabalhadores, a distância entre duas soluções (x e x') aumenta quando um trabalhador é trocado de máquina, uma vez que ao trocar dois trabalhadores de máquina o tempo das operações O_{ij} são

Tabela 2: Resultados (média do makespan) obtidos para cada implementação do PS.

Peso	Proximity Search		
	PS_1	PS_2	PS_{2RINS}
1	9802,4	**9468,4**	9305,8
10	9756,9	9502,5	**9274,3**
50	**9752,0**	9492,3	9276,0
100	9809,2	9566,7	9311,3

alterados. Com os pesos na função de proximidade o PS é forçado a permutar com mais frequência as tarefas ao invés de permutar os trabalhadores.

5.1. Resultados Computacionais

Para avaliar os resultados encontrados pelas implementações do PS e o MODELO, o valor de $RDP(\%)$ (Desvio Percentual Relativo) é calculado através da Equação (16), onde $Makespan$ é o valor do makespan econtrado pelo método em questão e ub é o melhor valor conhecido para determinada instância. As instâncias propostas por [Benavides et al. 2014] são baseadas nas instâncias Het-FSSP (Heterogeneous Flow Shop Scheduling Problem) de Carlier [Carlier 1978] e nas instâncias de flow shop scheduling de Taillard [Taillard 1993], sendo assim, os valores do ub para essas instâncias são encontrados em [Taillard 2004] e [Carlier e Pinson 1990].

$$RDP(\%) = \frac{Makespan - ub}{ub} \times 100 \quad (16)$$

Para análise dos resultados encontrados para as instâncias de Taillard, grupos de dez instâncias são formados e nesses casos a média é aparesentada como makespan final.

As Tabelas 3, 4, 6 e 7 apresentam os valores do $RDP(\%)$ para cada instância e cada método. Sendo que nas implementaçãos do PS utilizam-se os pesos que obtiveram menor média do makespan para todas as instâncias (como mostrado na Tabela 5). Os melhores valores encontrados para $RDP(\%)$ estão destacados em negrito.

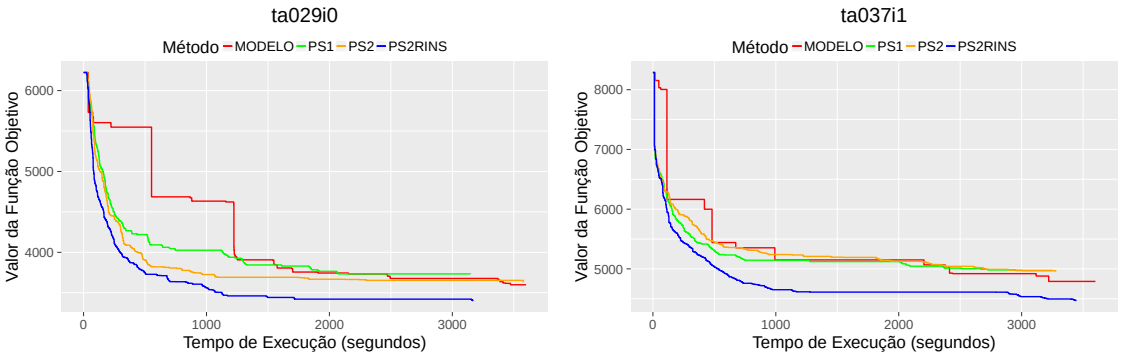
Para as instâncias de Carlier (consideradas pequenas e apresentado na Tabela 3 e Tabela 4) os resultados entre o MODELO e as implementações do PS são semelhantes, entretanto, o MODELO consegue ser melhor que o do PS em 6 instâncias, em contrapartida o PS é superior em 7 instâncias, ou seja, pelo menos uma das versões do PS consegue encontrar soluções melhores que o MODELO. Durante a execução do MODELO foram encontradas 13 soluções ótimas (em negrito), por conseguinte o PS conseguiu encontrar apenas 12 dessas soluções. A Tabela 5 apresenta o tempo gasto por cada método para encontrar a solução ótima, nesses casos pelo menos uma implementação do PS se mostra mais eficientes em relação ao tempo para encontrar a solução ótima.

Para as instâncias de Taillard (consideradas grandes e tem os resultados apresentados na Tabela 6 e Tabela 7) o PS_{2RINS} se sobressai melhor em 22 grupos, o PS_2 é melhor em 3 grupos e o MODELO tem vantagem sobre 11 grupos, no total de 36 grupos. Entretanto nos 3 grupos em que o PS_2 é melhor, o PS_{2RINS} é o método que possui o segundo melhor $RDP(\%)$. Para estes grupos pode-se afirmar que independente das duas implementações do PS que utilizar, o resultado obtido é melhor que o encontrado pela resolução do modelo matemático utilizando o software IBM ILOG CPLEX. Além disso, o MODELO após 3600 segundos não consegue encontrar solução viável para 18 instâncias, contudo todas as implementações do PS conseguiram encontrar soluções viáveis para todas as instâncias rapidamente (nos piores casos o tempo médio gasto para o PS encontrar uma solução viável é de 600 segundos).

Como já citado anteriormente a principal característica do PS é encontrar uma quantidade de soluções melhoradas logo nos primeiros instantes de busca. Para comprovar essa característica, foram selecionadas 2 instâncias do conjunto de Taillard. A cada melhoria obtida pelo PS e MODELO foram armazenados os tempos gastos para obter a solução. Ao final foi gerado um gráfico $tempo(s)$ X $Makespan$ para cada instância. Os resultados são apresentados na Figura 3.

Tabela 3: RDP(%) encontrado pelo $MODELO$, PS_1, PS_2 e PS_{2RINS} para as instâncias de Carlier com tempo de processamento entre t e $2t$.

instância	MxN	ub	Valor de $RDP(\%)$			
			MODELO	PS_1	PS_2	PS_{2RINS}
car1i0.txt	11x5	9952	**0,00**	**0,00**	**0,00**	0,26
car1i1.txt	11x5	9952	**0,00**	**0,00**	**0,00**	0,14
car1i2.txt	11x5	9952	**0,00**	**0,00**	**0,00**	**0,00**
car2i0.txt	13x4	10224	**0,00**	**0,00**	**0,00**	**0,00**
car2i1.txt	13x4	10224	**0,00**	**0,00**	**0,00**	**0,00**
car2i2.txt	13x4	10398	**0,00**	**0,00**	**0,00**	**0,00**
car3i0.txt	12x5	10268	1,26	**1,19**	1,26	2,44
car3i1.txt	12x5	10359	0,37	1,54	**0,00**	0,37
car3i2.txt	12x5	10359	0,37	**0,00**	1,54	**0,00**
car4i0.txt	14x4	11613	**0,00**	**0,00**	2,63	**0,00**
car4i1.txt	14x4	11846	**0,00**	**0,00**	0,70	**0,00**
car4i2.txt	14x4	11876	**0,13**	0,99	1,91	1,91
car5i0.txt	10x6	10589	**0,00**	1,84	0,34	0,34
car5i1.txt	10x6	10589	**0,00**	**0,00**	4,04	**0,00**
car5i2.txt	10x6	10848	**0,00**	**0,00**	**0,00**	**0,00**
car6i0.txt	8x9	11044	**0,00**	0,24	0,45	**0,00**
car6i1.txt	8x9	11044	**0,00**	**0,00**	**0,00**	**0,00**
car6i2.txt	8x9	11118	**0,00**	1,74	1,56	**0,00**
car7i0.txt	7x7	8558	**0,00**	**0,00**	**0,00**	**0,00**
car7i1.txt	7x7	8558	**0,00**	**0,00**	**0,00**	**0,00**
car7i2.txt	7x7	8558	**0,00**	**0,00**	**0,00**	**0,00**
car8i0.txt	8x8	11533	**0,36**	1,61	**0,36**	1,09
car8i1.txt	8x8	11659	**0,00**	0,17	0,79	0,72
car8i2.txt	8x8	11742	**0,00**	**0,00**	**0,00**	**0,00**
Médias Totais		10535,96	**0,10**	0,39	0,65	0,30



(a) Instância com m = 10 e n = 20

(b) Instância com m = 20 e n = 20

Figura 3: Atualização da solução de diferentes instâncias ao decorrer de 3600 segundos (Tempo(s) X Makespan) usando o $MODELO$, PS_1, PS_2 e PS_{2RINS}.

Tabela 4: RDP(%) encontrado pelo $MODELO$, PS_1, PS_2 e PS_{2RINS} para as instâncias de Carlier com tempo de processamento entre t e $5t$.

instância	MxN	ub	Valor de $RDP(\%)$			
			MODELO	PS_1	PS_2	PS_{2RINS}
car1I0.txt	11x5	19508	**0,00**	**0,00**	**0,00**	**0,00**
car1I1.txt	11x5	19831	**0,00**	**0,00**	**0,00**	**0,00**
car1I2.txt	11x5	19831	**0,00**	**0,00**	**0,00**	**0,00**
car2I0.txt	13x4	19876	**0,00**	0,08	0,08	0,01
car2I1.txt	13x4	19876	**0,00**	**0,00**	**0,00**	**0,00**
car2I2.txt	13x4	19876	**0,00**	**0,00**	1,37	1,37
car3I0.txt	12x5	19498	0,29	0,29	0,29	**0,00**
car3I1.txt	12x5	19498	**0,29**	1,66	5,08	**0,29**
car3I2.txt	12x5	19498	0,29	**0,00**	0,29	0,29
car4I0.txt	14x4	20381	0,20	**0,00**	0,20	**0,00**
car4I1.txt	14x4	22270	**0,00**	1,03	1,03	1,03
car4I2.txt	14x4	22270	**0,00**	**0,00**	0,05	0,34
car5I0.txt	10x6	18693	**0,00**	1,05	**0,00**	**0,00**
car5I1.txt	10x6	18693	**0,00**	**0,00**	**0,00**	**0,00**
car5I2.txt	10x6	19468	**0,00**	4,56	**0,00**	**0,00**
car6I0.txt	8x9	20070	0,34	6,74	**0,00**	**0,00**
car6I1.txt	8x9	20070	**0,00**	2,57	3,18	6,39
car6I2.txt	8x9	20070	**0,00**	3,13	2,18	6,58
car7I0.txt	7x7	16423	**0,00**	**0,00**	**0,00**	**0,00**
car7I1.txt	7x7	17067	**0,00**	**0,00**	0,32	**0,00**
car7I2.txt	7x7	17257	**0,00**	**0,00**	**0,00**	**0,00**
car8I0.txt	8x8	19159	**0,00**	**0,00**	0,92	**0,00**
car8I1.txt	8x8	19159	**0,00**	1,20	0,92	0,92
car8I2.txt	8x8	19791	**0,00**	2,74	**0,00**	**0,00**
Médias Totais		19505,54	**0,06**	1,04	0,66	0,72

Tabela 5: Tempo gasto pelos métodos $MODELO$, PS_1, PS_2 e PS_{2RINS} para encontrar a solução ótima.

Instância	ub	Tempo em segundos			
		MODELO	PS_1	PS_2	PS_{2RINS}
car1i0.txt	9952	486	41	**14**	-
car1i2.txt	9952	75	67	85	**16**
car6i1.txt	11044	2685	1577	673	**45**
car7i0.txt	8558	30	36	**20**	34
car7i1.txt	8558	13	41	33	**6**
car7i2.txt	8558	12	**4**	9	872
car5I0.txt	18693	168	-	249	**26**
car6I2.txt	20070	1696	-	-	-
car7I0.txt	16423	15	189	646	**2**
car7I1.txt	17067	590	**70**	-	3591
car7I2.txt	17257	46	**2**	20	3
car8I0.txt	19159	500	30	-	**29**
car8I2.txt	19791	1272	-	29	**6**

Tabela 6: RDP(%) encontrado pelo $MODELO$, PS_1, PS_2 e PS_{2RINS} para as instâncias de Taillard com tempo de processamento entre t e $2t$.

instância	MxN	ub	Valor de $RDP(\%)$			
			MODELO	PS_1	PS_2	PS_{2RINS}
ta001i0–ta010i0	20x5	1741,00	4,95	8,89	7,00	**4,74**
ta001i1–ta010i1	20x5	1765,70	**3,09**	8,42	8,34	6,36
ta001i2–ta010i2	20x5	1769,60	**3,71**	7,08	9,58	6,35
ta011i0–ta020i0	20x10	2103,40	17,44	18,71	14,21	**9,31**
ta011i1–ta020i1	20x10	2178,90	9,71	13,75	10,95	**6,13**
ta011i2–ta020i2	20x10	2180,00	13,43	15,42	9,42	**5,22**
ta021i0–ta030i0	20x20	3037,60	29,69	27,33	21,98	**11,27**
ta021i1–ta030i1	20x20	3141,40	37,69	20,46	15,17	**5,74**
ta021i2–ta030i2	20x20	3124,10	29,21	23,24	19,62	**7,17**
ta031i0–ta040i0	50x5	4007,60	**17,48**	46,80	40,46	34,58
ta031i1–ta040i1	50x5	4034,50	**19,30**	42,77	38,09	33,77
ta031i2–ta040i2	50x5	4037,80	**17,53**	38,02	36,51	27,51
ta041i0–ta050i0	50x10	4358,40	78,06	74,11	72,87	**65,76**
ta041i1–ta050i1	50x10	4499,00	64,31	58,42	60,36	**50,71**
ta041i2–ta050i2	50x10	4500,80	72,63	76,76	77,89	**69,90**
ta051i0–ta060i0	50x20	5565,20	198,35	185,35	176,26	**172,83**
ta051i1–ta060i1	50x20	5344,10	201,31	207,25	202,29	**191,42**
ta051i2–ta060i2	50x20	5335,20	193,91	181,91	176,56	**167,92**
Médias Totais		3484,68	56,21	58,59	55,42	**48,71**

Tabela 7: RDP(%) encontrado pelo $MODELO$, PS_1, PS_2 e PS_{2RINS} para as instâncias de Taillard com tempo de processamento entre t e $5t$.

instância	MxN	ub	Valor de $RDP(\%)$			
			MODELO	PS_1	PS_2	PS_{2RINS}
ta001I0–ta010I0	20x5	3343,70	**3,78**	13,29	7,79	8,25
ta001I1–ta010I1	20x5	3356,20	**4,24**	11,26	8,37	6,74
ta001I2–ta010I2	20x5	3375,20	**4,81**	9,37	7,31	8,24
ta011I0–ta020I0	20x10	3995,00	18,79	20,35	16,55	**11,32**
ta011I1–ta020I1	20x10	4086,70	16,17	20,21	15,23	**11,60**
ta011I2–ta020I2	20x10	4120,50	16,73	19,82	19,62	**9,74**
ta021I0–ta030I0	20x20	5796,20	49,37	30,06	24,66	**12,56**
ta021I1–ta030I1	20x20	5930,70	37,74	36,15	22,76	**10,40**
ta021I2–ta030I2	20x20	5934,60	30,42	36,02	25,41	**12,89**
ta031I0–ta040I0	50x5	7875,90	**21,75**	34,64	35,68	24,64
ta031I1–ta040I1	50x5	7923,80	**21,31**	37,45	29,61	22,72
ta031I2–ta040I2	50x5	7936,80	**19,85**	34,97	36,25	25,60
ta041I0–ta050I0	50x10	8488,10	87,27	83,20	80,31	**74,85**
ta041I1–ta050I1	50x10	8769,30	88,25	88,27	83,74	**82,29**
ta041I2–ta050I2	50x10	8777,80	87,92	83,49	82,29	**75,77**
ta051I0–ta060I0	50x20	11152,50	204,81	145,93	**125,79**	138,90
ta051I1–ta060I1	50x20	11055,11	220,98	152,95	**137,93**	145,10
ta051I2–ta060I2	50x20	11084,44	283,96	151,99	**136,49**	143,81
Médias Totais		6833,48	67,68	56,08	49,77	**45,86**

Como pode ser observado na Figura 3, as versões do PS conseguem encontrar soluções melhores que o MODELO com menor tempo de execução, e em muitos casos o valor encontrado pelo PS é melhor que o MODELO ao final dos 3600 segundos, sendo o PS_{2RINS} o método que obteve o melhor desempenho. O PS, de modo geral, consegue melhorar a solução, em muitas vezes antes dos 1800 segundos a solução encontrada pelo PS já é melhor que as soluções encontradas pelo MODELO.

6. Conclusão

De acordo com os experimentos realizados na seção anterior, para as instâncias pequenas, como as de Carlier, ambos os métodos utilizados para encontrar as soluções são equivalentes, entretanto o MODELO ainda possui uma pequena vantagem uma vez que conseguiu encontrar um numero maior de soluções ótimas, entretanto o tempo gasto pelo MODELO é muito maior se comparado com as implementações do PS para encontrar essas soluções. Para as instâncias grandes (Taillard) o PS_{2RINS} obteve melhor $RDP(\%)$ para grande parte das instâncias testadas superando as demais implementações do PS e o MODELO, se mostrando a melhor opção para a resolução desse problema.

Entre as versões do PS, de modo geral a versão PS_{2RINS} se sobressai sobre as outras, conseguindo soluções melhores em um menor espaço de tempo, como visto na Figura 3. Em relação ao tempo gasto pelos métodos, as implementações do PS têm vantagem sobre o MODELO, uma vez que conseguem encontrar e/ou melhorar as soluções logo no início da busca. Isso comprova a eficiência do PS sobre o MODELO para resolução do problema de flow shop não permutacional com trabalhadores heterogêneos em um curto espaço de tempo. Além disso, o *Proximity Search* se mostra um método promissor não só para as resoluções de problemas multicomponentes, mas também para a resolução de problemas que possuem um grande espaço de busca.

Como trabalhos futuros, os autores sugerem a elaboração de um sistema para determinar os pesos dinâmicos a serem aplicados as variáveis binárias de decisão utilizados na função proximidade do PS. Além disso avaliar o parâmetro utilizado pela heurística RINS que define o intervalo de nós da árvore de busca onde será aplicada a heurística. Neste trabalho foi adotado que a heurística RINS é aplicada em cada nó da árvore.

7. Agradecimentos

Os autores agradecem à CAPES, CNPq, FAPEMIG e a Divisão de Apoio ao Desenvolvimento Científico e Tecnológico da Universidade Federal de Viçosa.

Referências

Benavides, A. J., Ritt, M. e Miralles, C. (2014). Flow shop scheduling with heterogeneous workers. *European Journal of Operational Research*, 237(2):713–720.

Bonyadi, M. R., Michalewicz, Z. e Barone, L. (2013). The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, p. 1037–1044. IEEE.

Brasil (2010). Secretaria de direitos humanos presidência da república. `http://www.sdh.gov.br/assuntos/pessoa-com-deficiencia/dados-estatisticos/pesquisas-demograficas`. Acessado: 2016-04-10.

Brucker, P., Heitmann, S. e Hurink, J. (2003). Flow-shop problems with intermediate buffers. *OR Spectrum*, 25(4):549–574.

Carlier, J. (1978). Ordonnancements a contraintes disjonctives. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 12(4):333–350.

Carlier, J. e Pinson, E. (1990). A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26.

Danna, E., Rothberg, E. e Le Pape, C. (2005). Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90.

Fischetti, M. e Monaci, M. (2014). Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731.

Garey, M. R., Johnson, D. S. e Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129.

Liao, C., Liao, L. e Tseng, C. (2006). A performance evaluation of permutation vs. non-permutation schedules in a flowshop. *International Journal of Production Research*, 44(20):4297–4309.

Lin, S.-W. e Ying, K.-C. (2009). Applying a hybrid simulated annealing and tabu search approach to non-permutation flowshop scheduling problems. *International Journal of Production Research*, 47(5):1411–1424.

Potts, C. N. e Strusevich, V. A. (2009). Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society*, p. S41–S68.

Rossi, A. e Lanzetta, M. (2013). Scheduling flow lines with buffers by ant colony digraph. *Expert Systems with Applications*, 40(9):3328–3340.

Ruiz, R. e Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285.

Taillard, E. (2004). Scheduling instances. `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html`. Acessado: 2016-04-05.

Tandon, M., Cummings, P. e LeVan, M. (1991). Flowshop sequencing with non-permutation schedules. *Computers & chemical engineering*, 15(8):601–607.

Yagmahan, B. e Yenisey, M. M. (2010). A multi-objective ant colony system algorithm for flow shop scheduling problem. *Expert Systems with Applications*, 37(2):1361–1368.

Ying, K.-C. (2008). Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic. *The International Journal of Advanced Manufacturing Technology*, 38(3-4):348–354.