UMA ESTRATÉGIA DE BUSCA LOCAL EFICIENTE PARA O PROBLEMA DE MINIMIZAÇÃO DO ATRASO TOTAL RESULTANTE DO SEQUENCIAMENTO DA PRODUÇÃO EM UMA MÁQUINA COM TEMPOS DE SETUP

Katyanne Farias, Arthur Kramer, Anand Subramanian
Universidade Federal da Paraíba
Centro de Tecnologia, Campus I - Bloco G, Cidade Universitária, 58051-970, João Pessoa – PB
{katyannefaraujo, arthurhfrk}@gmail.com, anand@ct.ufpb.br

RESUMO

Esse trabalho trata do sequenciamento da produção em uma máquina com o objetivo de minimizar o atraso total. Cada tarefa está associada a um tempo de processamento e uma data de entrega. Para cada par de tarefas existe um tempo de *setup* s_{ij}, caso a tarefa j seja processada imediatamente após i. Este trabalho apresenta uma estratégia de limitação simples e eficiente para acelerar a busca local sem perda significativa na qualidade da solução. Essa estratégia consiste em um mecanismo de filtro que impede a avaliação de movimentos não promissores. Tal estratégia é implementada em uma meta-heurística baseada em busca local. Os experimentos computacionais revelam que a estratégia permite à meta-heurística ser altamente competitiva quando comparada à outros algoritmos da literatura, uma vez que é possível utilizar um grande número de vizinhanças, contribuindo para que o algoritmo alcance soluções de alta qualidade em um baixo tempo computacional.

PALAVRAS CHAVE. Sequenciamento, *Setup*, Busca Local

Área Principal: Meta-heurísticas, PO na Administração e Gestão da Produção

ABSTRACT

This work concerns the single machine production scheduling with the objective of minimizing the total tardiness. Each job has an associated processing time and due date. For each pair of jobs, there is a setup time s_{ij} in case job j is processed immediately after i. This work presents a simple yet effective limitation strategy that speeds up the local search procedure without a significant loss in the solution quality. Such strategy consists of a filtering mechanism that prevents unpromising moves to be evaluated. The proposed strategy has been embedded in a local search based metaheuristic. Computational experiments reveal that the strategy enables the metaheuristic to be extremely competitive when compared to other algorithms from the literature, since it allows the use of a large number of neighborhoods, contributing for the algorithm to achieve high quality solutions in a small computational time.

KEYWORDS. Scheduling, Setup, Local Search

Main Area: Metaheuristics, OR in Administration & Production Management

1. Introdução

Este artigo trata do problema de sequenciamento da produção em uma máquina, com tempos de *setup* e possibilidade de atraso. Pode-se defini-lo da seguinte forma: sendo $J = \{1, ..., n\}$ o conjunto de tarefas a serem sequenciadas em uma máquina, onde só pode ser processada apenas uma tarefa por vez e cada tarefa $j \in J$ possui um tempo de processamento p_j e um prazo de entrega d_j. Além disso, são considerados tempos de preparação, chamados de *setup*, em que s_{ij} é o tempo de *setup* necessário antes do processamento da tarefa $j \in J$ caso ela seja sequenciada imediatamente depois de $i \in J$. O objetivo é minimizar o atraso total $\sum T_j$, onde T_j para cada $j \in J$ está associado ao tempo de término C_j e é igual a $\max(C_j - d_j, 0)$, ou seja, assume valor positivo, caso a tarefa j seja concluída em atraso, e zero, caso contrário. Baseado na nomenclatura proposta por Graham et al. (1979), esse problema pode ser representado por $1|s_{ij}|\sum T_j$.

Solucionar um problema de sequenciamento pode ser bastante complexo, requerendo tipicamente um elevado esforço computacional. O problema $1||\sum T_j$, quando não são considerados tempos de *setup*, é comprovadamente tido como $\mathcal{NP}$-difícil, de acordo com Du e Leung (1990). Dessa forma, o problema $1|s_{ij}|\sum T_j$ também é tido como $\mathcal{NP}$-difícil por incluir como caso especial o problema $1||\sum T_j$.

Meta-heurísticas baseadas em busca local têm, em muitos casos, resultados satisfatórios para o problema abordado. Entretanto, a utilização desses métodos possui como um dos principais fatores limitantes o procedimento de avaliação de um movimento devido ao seu custo computacional, o qual, quando avaliado de maneira direta, possui complexidade $O(n)$. Tipicamente, a complexidade de se enumerar e avaliar todos os movimentos de vizinhanças clássicas, como a inserção e a troca, é $O(n^3)$.

Este trabalho apresenta um método simples que acelera o procedimento de busca local, embora não reduza a complexidade, baseado numa estimativa do quão vantajoso ou não é avaliar determinado movimento. Para o cálculo dessa estimativa, deve-se utilizar uma métrica capaz de ser computada rapidamente. O tempo normalmente dispendido na busca local pode limitar o número de vizinhanças utilizadas. A rapidez desse método viabiliza o uso de um número maior de vizinhanças, o que pode levar a soluções de alta qualidade em um tempo computacional aceitável. Para avaliar a eficiência do método de busca local desenvolvido será utilizado o algoritmo ILS-RVND proposto por Subramanian (2012).

Este trabalho está organizado como segue. A Seção 2 apresenta uma revisão dos principais algoritmos propostos na literatura para resolver o $1|s_{ij}|\sum T_j$. A Seção 3 descreve a estratégia de busca local, detalhando as vizinhanças utilizadas e o mecanismo de aceleração desenvolvido. Em seguida, a Seção 4 contém os resultados obtidos, comparando-os com os melhores disponíveis na literatura. Por fim, são apresentadas as conclusões na Seção 5.

2. Revisão da Literatura

Aplicações de diversos métodos (meta-)heurísticos são encontradas para a resolução do $1|s_{ij}|\sum T_j$. Dentre os primeiros trabalhos está o de Rubin e Ragatz (1995) com a aplicação de um Algoritmo Genético, (GA, do inglês *Genetic Algorithm*). Na etapa de reprodução, foi utilizado um operador *crossover* que leva em consideração a ordem absoluta das tarefas do indivíduo. Como mecanismo de mutação, foi utilizado a troca de blocos de duas tarefas, denominado *Random Search Pairwise Interchange* (RSPI). Uma maior diversificação foi alcançada com a inclusão de sequências aleatórias, chamadas "imigrantes". Além disso, os autores desenvolveram instâncias para o problema tratado com 15, 25, 35 e 45 tarefas, oito exemplares para cada caso.

França, Mendes e Moscato (2001) desenvolveram um GA, acrescido da aplicação de um Algoritmo Memético (MA, do inglês *Memetic Algorithm*), combinando os pontos fortes da abordagem de seleção dos indivíduos mais adaptados com a intensificação da busca local. Tan e Narasimhan (1997) propuseram um algoritmo *Simulated Anneling* (SA) com a

utilização de dois meios para a geração da solução inicial: *better than randomly generated* (PWS) e a geração aleatória, em inglês *randomly generated* (RND) com troca de pares de tarefas para melhorar a solução encontrada.

Gagné, Price e Gravel (2002) implementaram um algoritmo Colônia de Formigas (ACO, do inglês *Ant Colony Optimization*) com duas modificações principais: o *Random Search Pairwise Interchange* (RSPI) e o 3-opt restrito, usado no Problema do Caixeiro Viajante (TSP, do inglês *Traveling Salesman Problem*). Os referidos autores criaram instâncias com 55, 65, 75 e 85 tarefas. Gagné, Gravel e Price (2005) propuseram um algoritmo híbrido multiobjetivo para o problema estudado, unindo as meta-heurísticas Busca Tabu (TS, do inglês *Tabu Search*) e Busca em Vizinhança Variável (VNS, do inglês *Variable Neighborhood Search*). Gupta e Smith (2006) elaboraram duas heurísticas para a resolução do problema proposto: uma variação do *Problem Space-Based Local Search* (PSBLS) e um algoritmo *Greedy Randomized Adaptive Search Procedure* (GRASP) com *Path Relinking* (PR), que incorpora uma busca local baseada em uma variação das heurísticas VNS e Descida em Vizinhança Variável (VND, do inglês *Variable Neighborhood Descent*). Arroyo, Nunes e Kamke (2009) utilizaram um método de construção do GRASP para a geração da solução inicial e um método *Iterated Local Search* (ILS) para refinar a solução. Ying, Lin e Huang (2009) propuseram uma heurística *Iterated Greedy* (IG).

Liao e Juan (2007) elaboraram um algoritmo ACO com a introdução de novos parâmetros para o rastro de feromônio inicial e para o ajuste do tempo de aplicação da busca local. Sioud, Gravel e Gagné (2012) desenvolveram um algoritmo híbrido, integrando um GA a conceitos de domínios das posições relacionados à programação de restrições, conceitos de algoritmos evolutivos multiobjetivo e a regra de transição proporcional advinda do método ACO. Kirlik e Oğuz (2012) aplicaram um algoritmo *General Variable Neighborhood Search* (GVNS). O método proposto utiliza uma meta-heurística VNS, combinado à utilização do VND. Xu et al. (2014) propuseram diferentes versões de Algoritmos Evolucionários Híbridos, em inglês *Hybrid Evolutionary Algorithms* (HEAs), com a combinação de duas estratégias de atualização da população e três operadores *crossover*, incluindo o *linear order crossover operator* (LOX), onde a versão de melhor desempenho foi denominada LOX⊕B.

Subramanian, Battarra e Potts (2014) aplicaram um método ILS, adaptando o algoritmo ILS-RVND aos problemas $1|s_{ij}|\sum w_jT_j$ e $1|s_{ij}|\sum T_j$. O algoritmo proposto pelos autores foi definido no presente trabalho como ILS-RVND$_{\text{SBP}}$ para evitar interpretações ambíguas. O procedimento de busca local utilizado foi o *Randomized Variable Neighborhood Descent* (RVND). Como operador de permutação foi usado um *Double-Bridge*, originalmente proposto para o TSP. Para melhor explorar o espaço de soluções, foi desenvolvida uma versão do método que aceita soluções com o mesmo valor da função objetivo da solução corrente; para evitar a ciclagem do algoritmo, uma lista tabu foi utilizada.

Um algoritmo exato *Branch-and-Bound* (B&B) foi proposto por Bigras, Gamache e Savard (2008). Entretanto, o método exato sugerido por Tanaka e Araki (2013), além de abranger uma gama maior de problemas, apresentou melhores resultados. Trata-se de um algoritmo baseado no método *Successive Sublimation Dynamic Programming* (SSDP).

Ressalta-se que Liao e Juan (2007), Kirlik e Oğuz (2012), Xu et al. (2014) e Subramanian, Battarra e Potts (2014) aplicaram os algoritmos desenvolvidos também ao problema $1|s_{ij}|\sum w_jT_j$. A estratégia de aceleração da busca local proposta neste trabalho possibilita a utilização de um maior número de vizinhanças. Observa-se que na grande maioria dos trabalhos descritos anteriormente, foram utilizadas as vizinhanças de inserção e troca.

3. Aspectos Metodológicos

Esta seção descreve detalhadamente a metodologia utilizada para realização da busca local. Em um primeiro momento, é exposta a estratégia que limita os movimentos,

acelerando o procedimento de busca local. Em seguida, são apresentadas as estruturas de vizinhança representadas por meio de blocos, bem como a maneira como são computados os custos para cada estrutura quando um movimento é realizado. Por fim, é apresentado como a abordagem proposta foi implementada no algoritmo ILS-RVND.

3.1. Estratégia de Limitação da Busca Local

A enumeração e avaliação de todos os possíveis movimentos de uma estrutura de vizinhança de uma determinada solução do $1|s_{ij}|\sum T_j$ normalmente requer um elevado tempo computacional, sendo um dos principais gargalos em algoritmos baseados em busca local. Liao, Tsou e Huang (2012) desenvolveram um mecanismo que avalia os movimentos das vizinhanças de troca, inserção e inversão em um tempo dado por $O(n^2 logn)$; entretanto, ele passa a ser vantajoso apenas em instâncias com cerca de 1000 tarefas ou mais, segundo os próprios autores. Os procedimentos de avaliação dos movimentos das estruturas de vizinhança expostas nas Seções 3.2.1 e 3.2.2 são simples e de fácil implementação; entretanto, por possuírem complexidade $O(n^3)$, optou-se por analisar apenas um subconjunto da totalidade de movimentos por meio de uma estratégia de limitação da busca local.

A estratégia de limitação proposta é fundamentada em um mecanismo simples e eficiente que avalia apenas algumas soluções consideradas promissoras que resultam de movimentos das estruturas de vizinhança. O critério de avaliação do movimento é baseado na variação do *setup*, podendo ser computado em tempo constante. Caso essa variação não ultrapasse um limite preestabelecido, o movimento é avaliado. Observou-se que movimentos que levam a um aumento considerável nos tempos de *setup* fornecem geralmente soluções de baixa qualidade, devido ao seu impacto no custo da função objetivo.

O valor limite da variação do *setup* para cada vizinhança é representado por $max\Delta s_v$, $\forall v \in \mathcal{N}$, onde $\mathcal{N}$ é o conjunto das vizinhanças existentes na etapa de busca local. Portanto, um movimento de uma vizinhança $v \in \mathcal{N}$ será avaliado apenas se a variação de *setup* associada a ele for menor que $max\Delta s_v$. Como as características das instâncias podem intervir na eficiência desse parâmetro, não foi atribuído a ele um valor fixo. O $max\Delta s_v$ foi estimado por meio de uma fase de aprendizagem executada nas primeiras iterações da busca local, a qual é realizada sem o mecanismo de limitação. Primeiramente, $max\Delta s_v$, $\forall v \in \mathcal{N}$, é inicializado com um número M suficientemente grande para garantir que todos os movimentos das vizinhanças sejam avaliados. Considere Δs_v, $\forall v \in \mathcal{N}$, como sendo uma lista composta por variações do tempo de *setup*, onde cada variação está associada a um movimento de melhora da vizinhança em questão. Dessa forma, essa lista será incrementada com um novo valor sempre que for encontrada uma solução melhor que a solução corrente.

Ao final da fase de aprendizagem, a lista Δs_v, $\forall v \in \mathcal{N}$, é ordenada de maneira crescente. Seja $\theta \in [0, 1]$ um parâmetro de entrada. O elemento da lista ordenada associado à posição $\lfloor \theta \cdot |\Delta s_v| \rfloor$ é escolhido como o parâmetro limitante $max\Delta s_v$, $\forall v \in \mathcal{N}$. Quanto maior for o θ, maior será o $max\Delta s_v$ e mais movimentos serão avaliados; consequentemente, mais conservadora será a política de limitação. Exemplificando o parâmetro exposto, considere $\theta = 0,85$ e a seguinte lista de variações de *setup* $\Delta s_{troca} = [-8, -6, -3, -1, 3, 5, 12, 15, 27, 33]$. Portanto, $\lfloor \theta \cdot |\Delta s_v| \rfloor = \lfloor 0,85 \cdot 10 \rfloor = 8$. O valor referente a posição 8 da lista é 15, ou seja, $max\Delta s_{troca} = 15$.

Salienta-se que a lista Δs_v deve conter uma amostra suficientemente grande, garantindo a representatividade dos dados e uma consequente estimação satisfatória do parâmetro. Por outro lado, uma quantidade excessiva de itens na lista pode implicar em um número considerável de iterações na fase de aprendizagem, afetando negativamente o desempenho do algoritmo em termos de tempo computacional. Dessa forma, a execução completa da busca local na fase de aprendizagem, ou seja, a aplicação do mesmo número de iterações, não é estritamente necessária, devendo-se buscar um equilíbrio entre a quantidade de dados e o tempo despendido para obtenção da lista Δs_v.

A estratégia de limitação apresentada pode ser facilmente incorporada em meta-heurísticas baseadas em busca local, como por exemplo no GRASP ou qualquer outra meta-heurísticas *multi-start*, em que pode-se considerar a fase de aprendizagem apenas para um determinado número de iterações preliminares antes da estratégia de limitação ser acionada. No caso das meta-heurísticas que alternam entre etapas de intensificação e diversificação sistematicamente, como o TS, VNS e ILS, podem ser escolhidos como critérios de parada para a fase de aprendizagem o número de chamadas para o procedimento de busca local ou ainda um limite de tempo. Em meta-heurísticas baseadas em busca populacional que possuem procedimentos de busca local para aprimorar as soluções, também pode ser implementado um mecanismo semelhante.

3.2. Estruturas de Vizinhança

As estruturas de vizinhança estão representadas por meio de blocos. Seja uma sequência arbitrária com n tarefas $\{\pi_0, \pi_1, \pi_2, ..., \pi_n\}$. Um bloco B pode ser definido como uma subsequência de tarefas consecutivas. Uma tarefa *dummy* $\pi_0 = 0$ é utilizada para considerar o *setup* $s_{0\pi_1}$ que antecede o processamento da primeira tarefa da sequência. A Figura 1 apresenta uma sequência com 11 tarefas (com a primeira tarefa *dummy*) divididas em quatro blocos: $B_0 = \{\pi_0\}$; $B_1 = \{\pi_1, \pi_2, \pi_3, \pi_4\}$; $B_2 = \{\pi_5, \pi_6, \pi_7\}$; e $B_3 = \{\pi_8, \pi_9, \pi_{10}\}$.

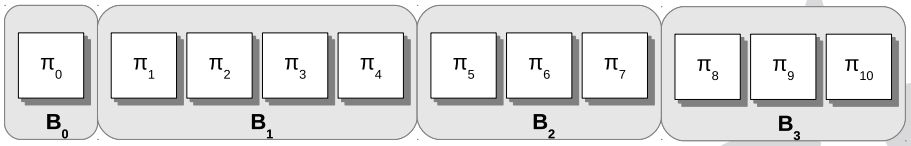

Figura 1: Exemplo de uma sequência dividida em 4 blocos

Sejam a e b as posições da primeira e da última tarefa do bloco B_t de uma sequência, respectivamente, e b' a posição da última tarefa do bloco B_{t-1} antecessor a B_t e que possui um tempo de término igual a C_b'. De acordo com o Algoritmo 1, o custo de B_t, ou seja, a soma dos atrasos ponderados de B_t, é computado em $O(|B_t|)$ etapas. Note que se $O(|B_t|) = 1$, as linhas 7-12 do Algoritmo 1 não serão executadas, já que nesse caso $b < a + 1$. Para fins ilustrativos, considerando o B_2 da Figura 1, tem-se $a = 5$, $b = 7$ e $b' = 4$. O cálculo do bloco considera desde a ligação entre B_1 e B_2 até o custo da última tarefa de B_2.

Algoritmo 1 CompCostBlock

```
1: Procedure CompCostBlock(b', a, b, π, Ctemp)
2: cost ← 0
3: Ctemp ← Ctemp + s_{π_{b'} π_a} + p_{π_a}        ▷ Variável global que armazena um tempo de término temporário
4: if Ctemp > d_{π_a} then
5:     cost ← Ctemp − d_{π_a}
6: end if
7: for a' = a + 1 ... b do
8:     Ctemp ← Ctemp + s_{π_{a'−1} π_{a'}} + p_{π_{a'}}
9:     if Ctemp > d_{π_{a'}} then
10:        cost ← cost + (Ctemp − d_{π_{a'}})
11:    end if
12: end for
13: return cost
14: end CompCostBlock.
```

Verifica-se que o custo total da sequência é obtido a partir do somatório dos custos de cada bloco pertencentes à sequência. No exemplo da Figura 1, o custo total é dado pela soma dos custos dos blocos B_0, B_1, B_2 e B_3. Para facilitar a avaliação de um movimento, pode-se utilizar uma estrutura de dados auxiliar que armazena o atraso ponderado acumulado até uma determinada posição da sequência. Dessa forma, optou-se por definir um vetor

g, onde g_j é a soma acumulada dos atrasos ponderados até uma tarefa j da sequência. Em seguida, são descritas as estruturas de vizinhanças utilizadas na abordagem implementada.

3.2.1. Troca

A estrutura de vizinhança troca consiste em permutar a posição de duas tarefas de uma sequência, conforme ilustrado na Figura 2. Observa-se que a realização de um movimento dessa vizinhança gera uma modificação na solução que pode ser representada por cinco blocos. Como o g_{i-1} e o $C_{\pi_{i-1}}$ são pré-computados, inicia-se o cálculo a partir do terceiro bloco, realizando-o em $O(n-i)$ passos, utilizando o Algoritmo 1.

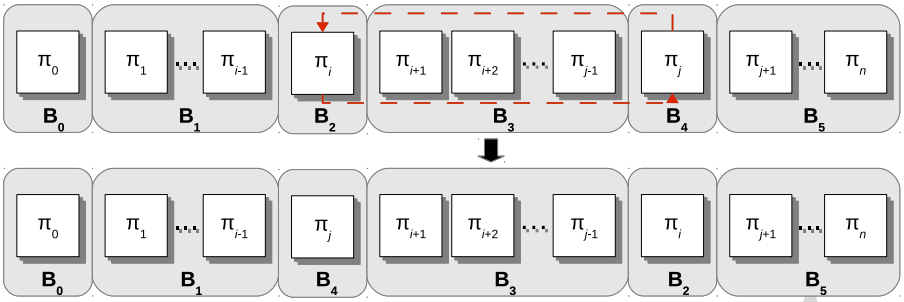

Figura 2: Troca de duas tarefas

3.2.2. Inserção l-Bloco

A estrutura de vizinhança inserção l-bloco consiste em mover uma subsequência de tamanho l de tarefas consecutivas. Considera-se i a posição da primeira tarefa do bloco e j a posição em que o bloco será inserido. Portanto, o bloco pode ser inserido em uma posição posterior ($i < j$), como ilustrado na Figura 3, ou anterior ($i > j$). Em ambos os casos, a solução resultante contém quatro blocos. Dessa forma, a avaliação do custo com a utilização do Algoritmo 1 é feita em $O(n-i)$ etapas para $i<j$; caso i seja maior do que j, o número de etapas do cálculo do custo é dado por $O(n-j)$.

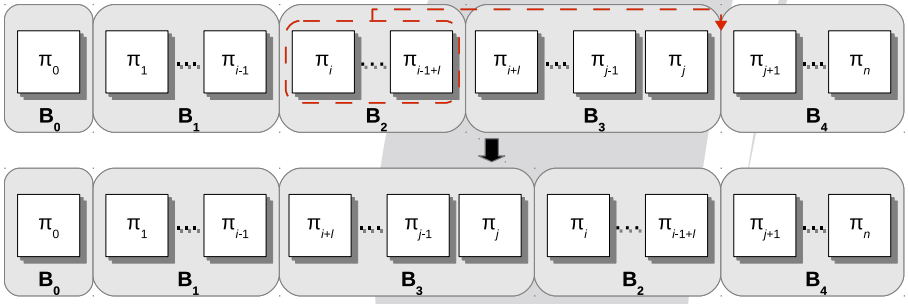

Figura 3: Inserção em posição posterior de um bloco de tamanho l

Sendo L um conjunto formado por diferentes valores positivos de l, pode-se definir uma única vizinhança com tamanhos de bloco que assumem qualquer valor de $l \in L$ ou definir múltiplas estruturas de vizinhança, onde cada uma está associada a um determinado valor de l. Essa última opção foi utilizada no método desenvolvido neste trabalho.

Vale ressaltar que foram feitos testes com a estrutura de vizinhança que realiza a troca de blocos de tarefas. Entretanto, essa estrutura não forneceu ganhos relevantes na qualidade da solução. Por consequência, a troca de blocos não foi utilizada na busca local.

3.3. Aplicação da Abordagem ao Algoritmo ILS-RVND

Essa seção descreve o ILS-RVND$_{Fast}$, desenvolvido a partir da heurística *multi-start* ILS-RVND proposta por Subramanian (2012). A principal diferença entre o método original ILS-RVND e o novo método proposto ILS-RVND$_{Fast}$ é a inclusão das etapas necessárias à incorporação da estratégia de limitação da busca local descritas na Seção 3.1. Optou-se por realizar a fase de aprendizagem necessária à implementação da estrutura de limitação na primeira iteração ILS do algoritmo, onde as listas Δs_v, $\forall v \in \mathcal{N}$ são incrementadas no procedimento de busca local; ao final da primeira iteração, o valor de $max\Delta s_v$ é estimado. Dois dos parâmetros de entrada de ambos os métodos são: I_R e I_{ILS}, os quais correspondem ao número de reinícios do método e o número de iterações ILS sem melhora consecutivas, respectivamente. No caso da fase de aprendizagem, apenas metade das iterações ILS são executadas, permitindo que as listas Δs_v concebidas no procedimento de busca local obtenham uma amostra satisfatória em um tempo computacional razoável.

Em ambos os algoritmos, a solução inicial é gerada por meio de uma heurística de inserção com componentes aleatórios e o mecanismo de perturbação usado é o *double-bridge*, que consiste na troca aleatória de dois blocos. Já a busca local utilizada é o procedimento RVND, o qual consiste em selecionar aleatoriamente uma vizinhança ainda não explorada sempre que outra vizinhança não encontrar uma solução melhor que a corrente. Em caso de melhoria da solução, todas as vizinhanças são consideradas aptas a serem exploras. O RVND possui como parâmetro L associado à vizinhança inserção l-bloco, devendo este ser calibrado, diferente do ILS-RVND$_{\text{SBP}}$, em que L é um conjunto fixo dado por $L = \{1, 2, 3\}$, referindo-se à utilização das vizinhanças inserção, inserção 2-bloco e inserção 3-bloco.

Um mecanismo de parada é utilizado quando encontrada uma sequência π com custo $f(\pi) = 0$. Nesse caso, como a solução ótima foi obtida, não é preciso continuar a busca. O ILS-RVND e o ILS-RVND$_{Fast}$ fazem uso desse mecanismo, embora o algoritmo proposto por Subramanian, Battarra e Potts (2014) não o tenha considerado.

4. Resultados Computacionais

Nesta seção são expostos os resultados obtidos pelo algoritmo ILS-RVND$_{Fast}$ para o problema $1|s_{ij}|\sum T_j$. O ILS-RVND$_{Fast}$ foi implementado na linguagem de programação C++, os experimentos foram realizados em um processador Intel Core i7 com 3,40 GHz e 16 GB de RAM e foram executados no Linux Mint 13. Apenas uma *thread* foi utilizada para a execução dos testes. As principais instâncias consideradas para a avaliação dos algoritmos propostos para o problema $1|s_{ij}|\sum T_j$ foram desenvolvidas por Rubin e Ragatz (1995) e Gagné, Price e Gravel (2002). Os autores elaboraram no total 64 instâncias divididas em grupos, onde as instâncias pertencentes a cada grupo possuem um determinado número n de tarefas que varia entre 15 e 85. Rubin e Ragatz (1995) criaram instâncias de 15 à 45 tarefas e Gagné, Price e Gravel (2002) utilizaram os mesmos parâmetros considerados pelos demais autores, gerando instâncias de 55 à 85 tarefas.

Para a determinação do número de iterações ILS, optou-se por utilizar o mesmo valor do encontrado no algoritmo ILS-RVND$_{\text{SBP}}$, ou seja, $I_{ILS} = 4n$, possibilitando analisar o impacto no tempo computacional causado pela estratégia de limitação proposta. Tendo sido definido o I_{ILS} e comprovada a rapidez do algoritmo, verificou-se que é vantajoso dobrar o número de reinícios quando se compara a qualidade da solução ao tempo computacional. Portanto, utilizou-se $I_R = 20$. Para a calibração dos parâmetros L e θ foi selecionado um conjunto de 26 instâncias escolhidas com base na dificuldade que o algoritmo ILS-RVND$_{\text{SBP}}$ apresentou para alcançar a solução ótima.

Especificamente para a calibração do L, foram realizados 34 testes com diferentes combinações, a primeira delas com $L = \{1, 2, 3, 4\}$, ou seja, vizinhanças do tipo inserção l-bloco com blocos de 1 até 4 tarefas. Nas demais combinações, é incrementada uma vizinhança l-bloco por vez até que $L = \{1, ..., 20\}$, além da utilização ou não da vizinhança

troca. Optou-se por inicializar os testes partindo-se do conjunto de estruturas de vizinhança de até 4 tarefas, já que os resultados do ILS-RVND$_{\text{SBP}}$ sugerem que a utilização de $L = 1, 2$ e 3 é o que garante a eficiência do algoritmo. A configuração adotada foi a $L = \{1, \ldots, 5\}$ + troca, pois esta aparenta fornecer soluções de qualidade satisfatória a um tempo computacional relativamente baixo. Para a calibração do parâmetro θ, foram testados valores variando entre 0,60 e 1,00 em intervalos de 0,05. Optou-se por utilizar $\theta = 0{,}75$, pois este valor gera uma diminuição significativa no tempo de execução sem, no entanto, afetar significativamente o custo das soluções.

Considere o *Melhor Gap* como sendo a diferença entre a melhor solução encontrada e a solução ótima. Contrariamente, o *Pior Gap* é a diferença entre a pior solução encontrada dentre as 10 execuções e a solução ótima. O *Gap Médio* é a diferença entre a média dos resultados das 10 execuções do algoritmo e a solução ótima. Nas tabelas posteriores, serão apresentadas as médias (aritmética ou geométrica) desses *Gaps* em porcentagem. Para as análises, o algoritmo foi executado 10 vezes para cada instância.

4.1. Impacto da Estratégia de Limitação

Como explicado na Seção 3.1, a estratégia utilizada limita a avaliação dos movimentos realizados na busca local. A Tabela 1 fornece a porcentagem média de movimentos que não foram avaliados na busca local por grupo de instâncias e por estrutura de vizinhança. Observa-se que uma média de no mínimo 83,8% dos movimentos são desconsiderados na análise feita no procedimento de busca local. Essa porcentagem aumenta proporcionalmente ao número de tarefas do grupo de instâncias, chegando a 95,0% quando n é 85.

Tabela 1: Porcentagem média dos movimentos que não foram avaliados $(1|s_{ij}|\sum T_j)$

Vizinhanças	n							
	15	25	35	45	55	65	75	85
Ins. 1-bloco	88,9	91,5	94,6	95,0	93,9	95,8	95,3	97,2
Ins. 2-bloco	85,9	93,9	94,6	94,4	93,4	94,2	93,8	95,6
Ins. 3-bloco	79,0	91,0	94,1	93,8	92,6	92,2	91,9	93,8
Ins. 4-bloco	78,9	90,2	92,4	94,0	93,1	92,3	91,5	93,4
Ins. 5-bloco	80,2	89,2	93,6	94,2	93,1	92,5	91,3	92,8
Troca	90,0	92,4	95,8	96,4	95,7	95,8	96,2	97,4
Média	83,8	91,4	94,2	94,6	93,6	93,8	93,3	95,0

A Tabela 2 apresenta a porcentagem média de movimentos que não foram avaliados e que levariam a soluções de qualidade superior por grupo e por vizinhança. Para obtenção destas informações, o algoritmo foi executado para todas as instâncias sem o uso da estratégia de limitação, verificando-se quando um movimento que levaria a uma solução de menor custo deixaria de ser avaliado caso o mecanismo de limitação fosse utilizado. Apesar de não avaliar boa parte dos movimentos como exposto anteriormente, para todos os grupos, a taxa máxima de movimentos relevantes que não foram avaliados é inferior a 18%.

Tabela 2: Porcentagem média dos movimentos que não foram avaliados e que levariam a melhores soluções $(1|s_{ij}|\sum T_j)$

Vizinhanças	n							
	15	25	35	45	55	65	75	85
Ins. 1-bloco	20,1	10,7	16,3	17,1	14,5	16,5	13,7	16,7
Ins. 2-bloco	17,0	9,9	15,7	17,4	13,1	18,9	15,7	16,8
Ins. 3-bloco	26,0	14,2	14,4	14,9	13,1	18,4	14,0	19,3
Ins. 4-bloco	27,2	15,3	25,0	19,1	21,2	20,1	20,8	19,9
Ins. 5-bloco	26,9	19,0	23,0	26,3	22,1	21,1	22,0	23,7
Troca	15,2	12,2	15,7	16,3	13,2	16,8	14,1	17,0
Média	22,1	13,5	18,3	18,5	16,2	18,6	16,7	18,9

Por fim, para confirmar a sua eficiência em termos de qualidade da solução e tempo de execução, foram comparados o ILS-RVND$_{Fast}$ com a versão deste algoritmo sem o uso da

estratégia de limitação (ILS-RVND). A Tabela 3 fornece os *Gaps* médios em porcentagem da melhor, pior e da média das soluções para os dois métodos por grupo e para o total de instâncias. A média geométrica para os grupos de 15 e 25 tarefas não pôde ser calculada, pois todos os *Gaps* de ambos os grupos são iguais a zero. Levando em consideração o total de instâncias, os *Gaps Médio* e *Pior* são semelhantes para ambos os métodos. Dessa forma, pode-se afirmar que a qualidade das soluções não é afetada de maneira perceptível. Em relação ao tempo de execução, o ILS-RVND$_{Fast}$ obteve uma média de 6,6 s, considerando todas as 64 instâncias disponíveis. Já o ILS-RVND obteve um tempo médio igual à 47,3 s. Percebe-se, portanto, uma redução de mais de 7 vezes no tempo computacional.

Tabela 3: Resultados do ILS-RVND vs resultados do ILS-RVND$_{Fast}$ $(1|s_{ij}|\sum T_j)$

		n								
		15	25	35	45	55	65	75	85	Total
Média Aritm. do Melhor *Gap* (%)	ILS-RVND	0,00	0,00	0,00	0,00	0,34	0,30	0,34	1,01	0,25
	ILS-RVND$_{Fast}$	0,00	0,00	0,00	0,00	0,00	0,00	0,11	0,61	0,09
Média Geom. do *Gap* Médio (%)	ILS-RVND	–	–	0,01	0,77	1,10	2,27	0,95	1,83	0,15
	ILS-RVND$_{Fast}$	–	–	0,09	0,78	1,26	1,22	0,99	1,60	0,12
Média Geom. do Pior *Gap* (%)	ILS-RVND	–	–	0,07	2,05	1,52	3,05	1,53	2,36	0,35
	ILS-RVND$_{Fast}$	–	–	0,20	1,71	2,09	1,64	1,37	2,24	0,34
Média Aritm. dos Tempos Médios (s)	ILS-RVND	0,1	0,5	3,7	11,1	22,2	54,3	90,6	195,9	47,3
	ILS-RVND$_{Fast}$	$< 0,1$	0,1	0,7	1,8	3,6	7,8	13,3	25,7	6,6

4.2. Comparação com a Literatura

Nesta seção são comparados os resultados do ILS-RVND$_{Fast}$, para as instâncias descritas anteriormente, aos dos principais algoritmos encontrados na literatura, a saber: ótimos obtidos pelo algoritmo exato proposto por Tanaka e Araki (2013) – **Opt**; Colônia de Formigas de Gagné, Price e Gravel (2002) – **ACO$_{GPG}$**, melhor de 20 execuções; Busca Tabu e Busca em Vizinhança Variável de Gagné, Gravel e Price (2005) – **Tabu$_{GGP}$**, melhor de 10 execuções; Colônia de Formigas de Liao e Juan (2007) – **ACO$_{LJ}$**, melhor de 10 execuções; *Iterated Greedy* de Ying, Lin e Huang (2009) – **IG**, melhor de 10 execuções; *General Variable Neighborhood Search* de Kirlik e Oğuz (2012) – **GVNS**, melhor de 20 execuções; *Iterated Local Search + Randomized Variable Neighborhood Descent* de Subramanian, Battarra e Potts (2014) – **ILS-RVND$_{SBP}$**, melhor, média e pior de 10 execuções; Algoritmo Evolucionário Híbrido de Xu et al. (2014) – **LOX⊕B**, melhor de 100 execuções.

A Tabela 4 apresenta uma comparação entre as soluções do ILS-RVND$_{Fast}$ com as dos principais algoritmos existentes para o problema tratado. Observa-se que todos os melhores resultados e médias obtidas por meio do ILS-RVND$_{Fast}$ apresentam qualidade superior aos demais. Salienta-se ainda que para a grande maioria das instâncias, a pior solução e a média encontrada pelo algoritmo possuem custo de valor igual ou inferior ao da melhor solução reportada pelos outros métodos. Além disso, as piores soluções foram ainda melhores ou iguais que todas as médias alcançadas por ILS-RVND$_{SBP}$. Esses resultados sugerem que o ILS-RVND$_{Fast}$ tem comparativamente melhor performance no que diz respeito à qualidade da solução.

A Tabela 5 apresenta o tempo médio, separado por grupos associados ao tamanho das instâncias, necessário para que o ILS-RVND$_{Fast}$ alcance as soluções dos principais métodos. São reportados também o tempo mínimo, máximo e a mediana. As duas instâncias com 85 tarefas que não tiveram a otimalidade comprovada por Tanaka e Araki (2013) não foram consideradas nessa análise, bem como a instância 751, a qual o algoritmo proposto não encontrou a solução ótima.

Tabela 4: Comparação entre o ILS-RVND$_{Fast}$ e os principais métodos heurísticos da literatura ($1|s_{ij}|\sum T_j$)

	ACO_{GPG}	$Tabu_{GGP}$	ACO_{LJ}	IG	GVNS	ILS-$RVND_{SBP}$	LOX⊕B
#*Melhores* aprimorados	36	29	32	19	18	13	5
#*Melhores* iguais	28	35	32	43	46	51	59
#*Melhores* piores	0	0	0	0	0	0	0
#*Médias* melhores que o melhor	36	26	32	18	15	6	1
#*Médias* iguais ao melhor	28	32	30	38	39	39	39
#*Médias* aprimoradas	–	–	–	–	–	27	–
#*Médias* iguais	–	–	–	–	–	37	–
#*Médias* piores	–	–	–	–	–	0	–
#*Piores* melhores que o melhor	36	22	30	14	7	1	0
#*Piores* iguais ao melhor	28	32	30	38	41	40	39
#*Piores* melhores que a média	–	–	–	–	–	12	–
#*Piores* iguais à média	–	–	–	–	–	36	–
#*Piores* aprimorados	–	–	–	–	–	11	–
#*Piores* iguais	–	–	–	–	–	33	–
#*Piores* piores	–	–	–	–	–	20	–
#valores reportados	64	64	64	62[1]	64	64	64

[1]: resultado da instância 751 não reportado e valor menor que ótimo na 703

Para as instâncias de até 45 tarefas, o maior tempo médio obtido foi de 1,3 segundos, além de possuir um tempo máximo sempre inferior à 5,2 s para alcançar os resultados atingidos pelos métodos citados, incluindo as soluções ótimas. A partir de 55 tarefas, os tempos médios de execução para chegar as melhores soluções dos algoritmos ACO_{GPG}, $Tabu_{GGP}$, ACO_{LJ}, IG e GVNS são respectivamente 0,1, 1,8, 3,4, 4,5 e 4,7 s. Quando considerados o ILS-$RVND_{SBP}$ e o LOX⊕B, esse tempo aumenta, passando a ser 19,4 e 35,6 s devido a qualidade superior das soluções relatadas pelos autores. Todavia, o tempo computacional do ILS-RVND$_{Fast}$ para as instâncias com mais de 45 tarefas é ainda consideravelmente inferior aos tempos relatados pelos autores de ambos os métodos: em média 52,83 s para o ILS-$RVND_{SBP}$; e um critério de parada de 100 s para o LOX⊕B. Portanto, conclui-se que além de apresentar resultados de qualidade similar ou superior, o ILS-RVND$_{Fast}$ é mais eficiente em termos de tempo de execução.

5. Considerações Finais

Este artigo apresentou uma estratégia eficiente de limitação da busca local para o $1|s_{ij}|\sum T_j$. O objetivo dessa estratégia é acelerar o procedimento de busca local, avaliando apenas os movimentos das estruturas de vizinhança que são considerados promissores. O critério utilizado para analisar a relevância dos movimentos é baseado na variação dos tempos de *setup*. Dessa forma, um movimento só é analisado se a variação de *setup* obtida a partir dele for menor que um valor limite, o qual depende das características da instância e da estrutura de vizinhança. O valor limite é definido em uma etapa de aprendizado e o grau de limitação da estratégia é determinado por um parâmetro de entrada do método. A consequente redução no tempo computacional possibilitou a utilização de um número maior de vizinhanças, fator visto como determinante para a boa performance do método.

A estratégia de limitação somada às estruturas de vizinhança utilizadas possibilitou ao ILS-RVND$_{Fast}$ obter soluções de alta qualidade em um tempo computacional relativamente baixo. Os experimentos computacionais realizados sugerem que o método proposto possui resultados melhores em termos de custo e tempo de execução quando comparados aos dos demais métodos da literatura. Em todas as instância do problema $1|s_{ij}|\sum T_j$, desconsiderando as duas instâncias em que a solução ótima não é comprovada, o ILS-RVND$_{Fast}$ alcançou a melhor solução obtida pelos principais métodos disponível na literatura, como exposto na Tabela 4. Devido a visível qualidade das soluções encontradas pelo algoritmo desenvolvido, pretende-se, além de investigar possíveis melhorias, aplicá-lo ao problema que considera máquinas paralelas. Outro possível trabalho futuro é o desenvolvimento de um

método híbrido para a resolução dos problemas tratados neste trabalho.

Tabela 5: Tempo médio em segundos despendido alcançar as soluções dos principais algoritmos da literatura $(1|s_{ij}|\sum T_j)$

			n								Total
			15	25	35	45	55	65	75	85	
ACO_{GPG}	Melhor	Média	< 0,1	< 0,1	< 0,1	0,1	0,1	0,5	0,1	0,2	0,1
		Mediana	< 0,1	< 0,1	< 0,1	< 0,1	0,1	0,1	0,1	0,2	< 0,1
		Mínimo	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1
		Máximo	< 0,1	< 0,1	0,1	0,2	0,2	2,9	0,5	0,6	2,9
$Tabu_{GGP}$	Melhor	Média	< 0,1	< 0,1	0,1	0,9	2,5	3,7	2,2	4,5	1,8
		Mediana	< 0,1	< 0,1	0,1	0,3	0,3	1,6	0,6	2,7	0,1
		Mínimo	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1
		Máximo	< 0,1	< 0,1	0,3	3,2	16,7	18,8	6,4	15,3	18,8
ACO_{LJ}	Melhor	Média	< 0,1	< 0,1	0,1	0,1	20,7	2,3	2,1	1,6	3,4
		Mediana	< 0,1	< 0,1	< 0,1	< 0,1	0,1	1,1	0,8	0,6	< 0,1
		Mínimo	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1
		Máximo	< 0,1	0,1	0,6	0,4	164,3	10,5	10,2	6,7	164,3
IG	Melhor	Média	< 0,1	< 0,1	0,2	0,7	21,3	4,0	5,3	3,8	4,5
		Mediana	< 0,1	< 0,1	0,2	0,5	0,6	2,4	6,9	4,3	0,1
		Mínimo	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1
		Máximo	< 0,1	< 0,1	0,4	3,1	164,3	14,8	12,5	7,3	164,3
GVNS	Melhor	Média	< 0,1	< 0,1	0,3	0,6	1,6	5,0	10,4	20,0	4,7
		Mediana	< 0,1	< 0,1	0,2	0,5	0,7	3,4	6,4	12,2	0,2
		Mínimo	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1
		Máximo	< 0,1	< 0,1	0,8	2,0	6,5	12,7	32,1	69,4	69,4
$ILS\text{-}RVND_{SBP}$	Melhor	Média	< 0,1	< 0,1	0,2	1,3	5,9	4,4	32,3	110,7	19,4
		Mediana	< 0,1	< 0,1	0,1	0,5	0,8	3,1	15,8	30,8	0,1
		Mínimo	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1
		Máximo	< 0,1	< 0,1	0,6	4,5	33,6	11,0	81,3	385,2	385,2
	Média	Média	< 0,1	< 0,1	0,2	0,4	1,0	3,4	7,4	16,5	3,6
		Mediana	< 0,1	< 0,1	0,1	0,4	0,7	1,9	5,3	9,6	0,1
		Mínimo	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1
		Máximo	< 0,1	< 0,1	0,5	0,9	2,7	10,9	24,4	53,2	53,2
LOX⊕B	Melhor	Média	< 0,1	< 0,1	0,2	1,0	25,3	51,4	129,4	77,2	35,6
		Mediana	< 0,1	< 0,1	0,2	0,4	0,5	11,8	118,5	26,6	0,1
		Mínimo	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1
		Máximo	< 0,1	< 0,1	0,8	3,4	186,6	311,3	315,3	265,6	315,3
Opt.[1]		Média	< 0,1	< 0,1	0,1	1,1	25,3	111,9	102,8	113,0	41,1
		Mediana	< 0,1	< 0,1	0,1	0,3	0,5	11,8	25,9	75,3	0,1
		Mínimo	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	< 0,1	0,1	< 0,1
		Máximo	< 0,1	< 0,1	0,5	5,2	186,6	503,5	247,9	359,7	503,5

[1]: Foram desconsideradas as instâncias 751 ($n = 75$), 851 e 855 ($n = 85$)

Referências

Arroyo, J. E. C.; Nunes, G. V. P.; Kamke, E. H. (2009), Iterative local search heuristic for the single machine scheduling problem with sequence dependent setup times and due dates. *IEEE Computer Society*, p. 505 – 510.

Bigras, L.-P., Gamache, M. e Savard, G. (2008), The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, v. 5, n. 4, p. 685 – 699.

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

França, P. M., Mendes, A. e Moscato, P. (2001), A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, v. 132, n. 1, p. 224 – 242.

Gagné, C., Price, W. L. e Gravel, M. (2002), Comparing an aco algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times.

The Journal of the Operational Research Society, v. 53, n. 8, p. 895 – 906.

Graham, R., Lawler, E., Lenstra, J. e Kan, A. Optimization and approximation in deterministic sequencing and scheduling: a survey. P.L. Hammer, E. J. e Korte, B. (Eds.), *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver, volume 5 of Annals of Discrete Mathematics*, p. 287 – 326. Elsevier, 1979.

Gupta, S. R. e Smith, J. S. (2006), Algorithms for single machine total tardiness scheduling with sequence dependent setups. *European Journal of Operational Research*, v. 175, n. 2, p. 722 – 739.

Kirlik, G. e Oğuz, C. (2012), A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. *Computers & Operations Research*, v. 39, n. 7, p. 1506 – 1520.

Lawler, E. L. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. P.L. Hammer, E.L. Johnson, B. K. e Nemhauser, G. (Eds.), *Studies in Integer Programming, volume 1 of Annals of Discrete Mathematics*, p. 331 – 342. Elsevier, 1977.

Lenstra, J., Kan, A. R. e Brucker, P. Complexity of machine scheduling problems. P.L. Hammer, E.L. Johnson, B. K. e Nemhauser, G. (Eds.), *Studies in Integer Programming, volume 1 of Annals of Discrete Mathematics*, p. 343 – 362. Elsevier, 1977.

Liao, C.-J. e Juan, H.-C. (2007), An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers & Operations Research*, v. 34, n. 7, p. 1899 – 1909.

Liao, C.-J., Tsou, H.-H. e Huang, K.-L. (2012), Neighborhood search procedures for single machine tardiness scheduling with sequence-dependent setups. *Theoretical Computer Science*, v. 434, p. 45 – 52.

Rubin, P. A. e Ragatz, G. L. (1995), Scheduling in a sequence dependent setup environment with genetic search. *Computers & Operations Research*, v. 22, n. 1, p. 85 – 99. Genetic Algorithms.

Sioud, A., Gravel, M. e Gagné, C. (2012), A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, v. 39, n. 10, p. 2415 – 2424.

Subramanian, A. *Heuristic, Exact and Hybrid Approaches for Vehicle Routing Problems*. Tese de doutorado, Universidade Federal Fluminense, Niterói, Brazil, 2012.

Subramanian, A., Battarra, M. e Potts, C. N. (2014), An iterated local search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, v. 52, n. 9, p. 2729 – 2742.

Tan, K. e Narasimhan, R. (1997), Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. *Omega*, v. 25, n. 6, p. 619 – 634.

Tanaka, S. e Araki, M. (2013), An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. *Computers & Operations Research*, v. 40, n. 1, p. 344 – 352.

Xu, H., Lü, Z., Yin, A., Shen, L. e Buscher, U. (2014), A study of hybrid evolutionary algorithms for single machine scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, v. 50, n. 0, p. 47 – 60.

Ying, K.-C., Lin, S.-W. e Huang, C.-Y. (2009), Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, v. 36, n. 3, Part 2, p. 7087 – 709.