TWTJSSP-ILS: UM ALGORITMO HEURÍSTICO PARA RESOLVER O PROBLEMA *JOB-SHOP SCHEDULING* COM PENALIDADE PELO TEMPO DE ATRASO

Raphael Carlos Cruz[1], Helena Ramalhinho Lourenço[2], Vitor Nazário Coelho[1] Marcone Jamilson Freitas Souza[1] e Alex Grasas[2]

[1]Departamento de Computação, Universidade Federal de Ouro Preto (UFOP)
Campus Universitário, Morro do Cruzeiro, CEP 35.400-000, Ouro Preto (MG), Brasil
raphaelcarlos25@yahoo.com.br, vncoelho@gmail.com, marcone@iceb.ufop.br

[2]Departamento de Economia e Negócios, Universidade Pompeu Fabra (UPF)
Ramon Trias Fargas, 27, CEP 08005, Barcelona (CAT), Espanha
helena.ramalhinho@upf.edu, alex.grasas@upf.edu

Abstract. *This work focuses on the Total Weighted Tardiness Job-Shop Scheduling Problem. In this problem, each job consists of a set of tasks that must be processed on a given set of machines for uninterrupted and predetermined time. Each job has a due date and a penalty associated with the delay in completion time. The objective is to minimize the total weighted tardiness. We propose an heuristic method based on Iterated Local Search (ILS) metaheuristic, designated as TWTJSSP-ILS. The method applies GRASP to obtain an initial solution, combining a random choice technique with the longest processing time. In the refinement stage of the ILS method, a Variable Neighborhood Descent (VND) procedure is applied as the as local search step. We present a computational experiment applying this method to the benchmark instances taken from the literature. The initial results lead to high quality solutions.*
KEYWORDS*: Total Weighted Tardiness Job-Shop Scheduling Problem, Iterated Local Search, Variable Neighborhood Descent.*

Resumo. *Este trabalho tem seu foco no problema de programação da produção do tipo Job-Shop Scheduling com penalidade pelo atraso em relação à data de entrega. No problema tratado, cada job consiste de um conjunto de tarefas que devem ser processadas em uma dada máquina durante um período de tempo ininterrupto e predeterminado. Cada job tem uma data de entrega e o objetivo é minimizar o atraso na conclusão da operação. Propõe-se o algoritmo heurístico TWTJSSP-ILS para resolvê-lo. Inicialmente uma solução é gerada utilizando o método GRASP, combinando uma técnica de escolha randômica e outra seguindo a regra do maior tempo de processamento. Na fase de refinamento, usa-se o método Iterated Local Search (ILS) que possui como busca local o procedimento Descida em Vizinhança Variável (VND). O algoritmo proposto foi testado em problemas-teste disponíveis na literatura e se mostrou capaz de gerar soluções de qualidade.*
PALAVRAS-CHAVE*: Job-Shop Scheduling com Penalidade pelo Tempo de Atraso, Iterated Local Search, Descida em Vizinhança Variável.*

1 Introdução

O seqüenciamento das atividades ou *scheduling* é uma das atividades que compõem o planejamento da produção. Nesta atividade são levados em consideração uma série de elementos que disputam vários recursos por um período de tempo, recursos esses que possuem capacidade limitada. Portanto, a maioria dos problemas de programação da produção estudados aplica-se ao ambiente conhecido como *Job-Shop*.

O problema *Job-Shop scheduling*, surgido no início da década de 50 (Jain e Meeran, 1999), consiste em sequenciar um conjunto de trabalhos num grupo de máquinas, minimizando uma determinada medida de performance, seja ela atendimento de prazos (ou datas de entrega), minimização do *makespan* (tempo de conclusão do processo), minimização do tempo de atraso em relação a entrega, minimização do tempo de fluxo dos estoques intermediários, maximização da utilização da capacidade disponível ou até mesmo a combinação destes objetivos (Walter, 1999).

A motivação para estudar métodos de solução para esse problema reside no fato de o mesmo ser largamente encontrado no cenário das indústrias manufatureiras, que cada vez mais procuram soluções eficientes para tornarem seus produtos mais competivos.

Existe na literatura um grande número de métodos exatos e heurísticos para resolver o problema *Job-Shop*, porém a grande maioria deles são desenvolvidos para tratarem do *makespan* como objetivo. Uma das abordagens mais bem sucedidas para este critério é o algoritmo proposto por Laarhoven et al. (1992) baseado no método *Simulated Annealing,* o procedimento Busca Tabu implementado por Nowicki e Smutnicki (1996) e o *Iterated Local Search* em Lourenço (1995). Algoritmos Genéticos também têm sido aplicados com sucesso para o problema *Job-Shop*, como é o caso de Volta et al. (1995). Uma abordagem relevante, que combina algoritmos genéticos com outras heurísticas de busca local, é proposta por Gonçalves et al. (2005).

Umas das variantes do *Job-Shop scheduling*, que tem como objetivo minimizar o tempo de atraso em relação à data de entrega do produto, é o *Total Weighted Tardiness Job-Shop Scheduling Problem* (TWTJSSP). Trata-se de um problema de sequenciamento de máquinas da classe NP-difícil; portanto, as principais abordagens presentes na literatura se baseiam em heurísticas, embora existam trabalhos que utilizam métodos exatos.

Pinedo e Singer (1998) apresentam e comparam uma série de algoritmos baseados na técnica *Branch and Bound* para minimizar o atraso total no problema *Job-Shop Scheduling*. Em Pinedo e Singer (1999), a heurística *Shifting Bottleneck* (SB) proposta por Adams et al. (1988), é utilizada com o intuito de decompor o problema em subproblemas menores com uma única máquina, sendo então resolvidos um após o outro, ou seja, cada máquina é programada de acordo com a solução de seu subproblema correspondente. Essa técnica se mostrou eficaz e produziu bons resultados.

Kreipl (2000) também aborda o problema de forma heurística utilizando uma técnica denominada de *large step random walk* que varia o tamanho da vizinhança de acordo com os passos seguidos pelo algoritmo. A técnica de algoritmos genéticos é combinada com o procedimento *Iterated Local Search* no trabalho proposto por Essafi et al. (2008), produzindo até então, umas das melhores soluções para o TWTJSSP. Bülbül (2011) propõe um algoritmo híbrido que contempla as heurísticas *Shifting Bottleneck* e Busca Tabu. Seu objetivo é particionar o problema, sendo a Busca Tabu responsável por realizar

a busca local e o método SB de diversificar as soluções.

Neste trabalho, o algoritmo denominado `TWTJSSP-ILS` incorpora o método de busca local *Variable Neighborhood Descent* – VND (Mladenović e Hansen, 1997) como busca local do método ILS (*Iterated Local Search*), além de uma estrutura auxiliar de dados que permite armazenar informações importantes ao longo da execução do algoritmo, contribuindo para fornecer suporte a um método rápido de avaliação.

O restante deste artigo está organizado como segue. Na seção 2 é descrito o problema de sequenciamento de máquinas do tipo *Job-Shop* com penalidade pelo tempo de atraso. Na seção 3 é apresentado o algoritmo proposto e na seção 4, os resultados dos experimentos computacionais. Na seção 5 o trabalho é concluído, assim como apontadas sugestões de trabalhos futuros.

2 Descrição do Problema

Descreve-se, a seguir, um problema da programação de produção do tipo *Job-Shop*, em sua forma básica. Um conjunto de *n jobs* J = $\{J_1, J_2, \ldots, J_n\}$ deve ser processado em um conjunto de *m* máquinas M = $\{M_1, M_2, \ldots, M_m\}$ disponíveis. Cada *job* possui uma ordem de execução específica entre as máquinas, ou seja, um *job* é composto de uma lista ordenada de tarefas (também conhecidas como operações), cada uma das quais definida pela máquina requerida e pelo tempo de processamento na mesma. As restrições que devem ser respeitadas são:

- As operações não podem ser interrompidas e cada máquina pode processar apenas uma tarefa de cada vez;
- Cada *job* só pode estar sendo processado em uma única máquina de cada vez;
- Cada *job* é executado por uma sequência conhecida de tarefas (operações).

Uma vez que as sequências de máquinas de cada *job* são fixas, o problema a ser resolvido consiste em determinar as sequências dos *jobs* em cada máquina, de forma que o tempo de execução transcorrido, desde o início da primeira tarefa até o término da última, não ultrapasse a data limite de término. Levando em consideração a abordagem deste trabalho, o objetivo é encontrar a solução com menor tempo de atraso para a execução dos *jobs*.

Na Figura 1 pode ser observado um exemplo de solução para o *Job-Shop Scheduling Problem* com *M=3* e *J=3*. A máquina 1, denotada por **M1** na parte superior da Figura 1, está executanto a primeira tarefa do *Job* 3, a primeira tarefa do *Job* 2 e por último, a segunda tarefa do *Job* 1 nesta ordem. Nota-se que a máquina 2 (**M2**) somente inicia a segunda tarefa do *Job* 2 quando a primeira tarefa deste mesmo *Job* é concluída na máquina 1 (**M1**). Logo, é possível perceber que todas as restrições do problema são atendidas.

3 Algoritmo Proposto

Nesta Seção é apresentado o algoritmo proposto para resolver o TWTJSSP, assim como os módulos que o compõe.

3.1 Algoritmo TWTJSSP-ILS

O algoritmo proposto, chamado de `TWTJSSP-ILS`, combina os procedimentos heurísticos GRASP, ILS e VND. O primeiro procedimento é usado para gerar uma solução

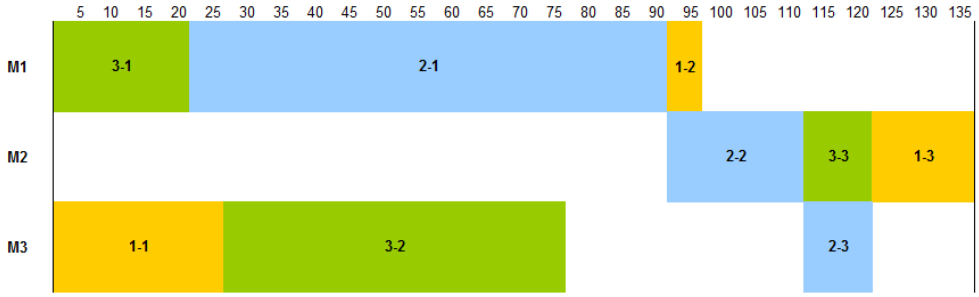

Figura 1. Exemplo de uma Solução para o problema *Job-Shop Scheduling*

inicial, sendo o refinamento feito pelo ILS. O ILS, por sua vez, utiliza o VND como módulo de busca local. Adicionalmente, ele utiliza uma estratégia de estrutura auxiliar de dados que permite armazenar informações importantes ao longo da execução do algoritmo, contribuindo para fornecer suporte a um método rápido de avaliação. Seu pseudocódigo é apresentado pelo Algoritmo 1.

Algoritmo 1: *TWTJSSP-ILS*

1: $\gamma \leftarrow$ número real aleatório no intervalo $[0.0, 0.7]$;
2: $s \leftarrow GRASP(\gamma, Time)$;
3: $s \leftarrow VND(s)$;
4: $iter \leftarrow 1$;
5: **enquanto** ($iter \leq maxIter$) **faça**
6: $s' \leftarrow Perturbação(s)$;
7: $s'' \leftarrow VND(s')$;
8: **se** ($f(s'') < f(s)$) **então**
9: $s \leftarrow s''$;
10: $iter \leftarrow 1$;
11: **senão**
12: $iter \leftarrow iter + 1$;
13: **fim se**
14: *Atualiza_Estrutura_Auxiliar_Dados(Job, Tarefa, Tempo, Máquina, Ordem)*;
15: **fim enquanto**
16: Retorne s;

Como pode ser observado no Algoritmo 1, o método GRASP descrito na Seção 3.3, produz a solução inicial s, que é em seguida entregue a um procedimento VND (Seção 3.5). A solução inicial gerada é então refinada pelo ILS. Para escapar das armadilhas de ótimos locais, e se dirigir para outras regiões do espaço de busca, foram utilizadas perturbações descritas na Seção 3.6. O método VND (descrito na Seção 3.5) é utilizado como busca local, sendo interrompido quando um número de iterações sem melhora na solução corrente (*maxIter*) é atingido.

3.2 Função de Avaliação

Uma solução é avaliada pela função (1), a ser minimizada. A primeira parcela do somatório, representada por W_i, é um fator de importância para a execução do *job*, ou seja, alguns *jobs* possuem prioridades em relação aos demais. Este fator é fornecido pela instância e varia de 1 a 4, sendo que 20% do primeiros *jobs* têm peso 4 e alta prioridade, 60% dos *jobs* subsequentes têm peso 2 e média prioridade e os 20% restantes têm peso 1 com baixa prioridade. A segunda parcela do somatório obtém o máximo valor entre 0 e

o tempo de atraso na execução do *job*. Intuitivamente, o tempo de atraso é definido pela diferença entre a data de conclusão C_i e a data de entrega d_i.

$$f(s) = \sum_{i \in J} W_i \cdot \max\{0, C_i - d_i\} \quad (1)$$

Na função de avaliação f, dada pela Equação (1), têm-se:
J: conjunto dos *jobs* a serem processados;
W_i: fator de importância do *job* $i \in J$;
C_i: Data de conclusão do *job* $i \in J$;
d_i: Data de entrega do *job* $i \in J$.

3.3 Geração da Solução Inicial

O algoritmo `TWTJSSP-ILS` gera uma solução inicial utilizando o método GRASP (Feo e Resende, 1995), o qual combina duas técnicas de escolha, uma randômica, onde a tarefa que vai ser inserida na máquina é escolhida aleatoriamente e outra seguindo a regra do maior tempo de processamento, onde é escolhida a tarefa candidata que possuir o maior tempo de execução entre o conjunto de tarefas candidatas da máquina. A busca local é realizada por meio do método Descida em Vizinhança Variável - VND (Mladenović e Hansen, 1997).

Inicialmente, um número real aleatório γ entre $0,0$ e $0,7$ é gerado para determinar o índice de aleatoriedade do GRASP. Outro parâmetro utilizado no método é o critério de parada, que neste caso é o tempo de execução ($time$).

A técnica de escolha randômica consiste em eleger uma tarefa de um determinado *job* de forma aleatória, verificar qual a máquina que deverá processar esta tarefa e por último alocá-la na máquina correspondente.

A regra gulosa que compõe o método GRASP se baseia em alocar as tarefas de acordo com o maior tempo de processamento que elas possuem. O primeiro passo é organizar as tarefas utilizando como critério a máquina onde serão executadas. Feita essa separação, deve-se ordenar essas tarefas em ordem decrescente de tempo, ou seja, do maior para o menor tempo de processamento. Por último, as tarefas são direcionadas para as máquinas correspondentes seguindo esta ordem.

3.4 Estruturas de vizinhança

Para explorar o espaço de soluções foram utilizados quatro tipos de movimentos: Troca, Realocação Monotarefa, Realocação Multitarefa e *2-Opt*. Todos eles são detalhados nas Seções 3.4.1, 3.4.2, 3.4.3 e 3.4.4, respectivamente.

3.4.1 Movimento de Troca (*Exchange*)

O Movimento de Troca, também chamado de *Exchange*, consiste em selecionar duas tarefas T_i e T_j, e em seguida trocá-las de posição, conforme ilustrado na Figura 2.

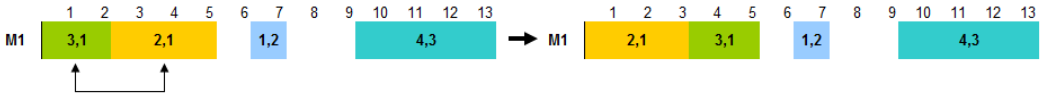

Figura 2. Exemplo da aplicação do Movimento de Troca (*Exchange*)

3.4.2 Movimento de Realocação Monotarefa

Para o Movimento de Realocação Monotarefa uma tarefa T_i da sequência é indicada, assim como a posição j para onde ela será realocada. Um exemplo para este movimento por ser visto na Figura 3.

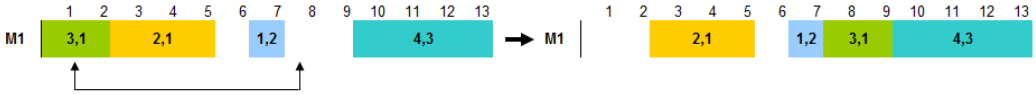

Figura 3. Exemplo da aplicação do Movimento de Realocação Monotarefa

3.4.3 Movimento de Realocação Multitarefa

O Movimento de Realocação Multitarefa é baseado na realocação de um bloco de k tarefas, com $2 \leq k \leq n-2$, sendo n o número total de tarefas de uma máquina. Portanto, um número k de tarefas (J_i, J_{i+1},...) são elegidas, assim como a posição j indicando a partir de onde elas serão alocadas. Um exemplo com $k = 2$ e posição $j = 8$ pode ser observado na Figura 4.

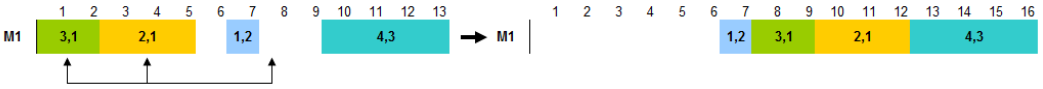

Figura 4. Exemplo da aplicação do Movimento de Realocação Multitarefa

3.4.4 Movimento *2-Opt*

O Movimento *2-Opt* consiste em remover dois arcos não adjacentes de forma que outros dois são adicionados e uma nova sequência de tarefas é gerada. Quando os arcos são removidos, novos arcos são formados com o intuito de preencher o espaço vazio ocasionado na remoção. A Figura 5 ilustra a remoção dos arcos indicados com um "x" e a solução final gerada com este movimento.

Figura 5. Exemplo da aplicação do Movimento *2-Opt*

3.5 VND

O método VND implementado é baseado na proposta de Mine et al. (2010), onde são utilizadas duas estratégias adicionais em relação ao método clássico. A primeira consiste em definir aleatoriamente a ordem das vizinhanças a serem exploradas. A outra estratégia consiste em intensificar a busca nas máquinas modificadas, isto é, aplicar uma

busca local do tipo *Best Improvement* com os quatro movimentos descritos na Subseção 3.4. O pseudocódigo do VND é apresentado no Algoritmo 2.

Algoritmo 2: VND

Seja r o número de estruturas distintas de vizinhanças;
$\mathcal{RN} \leftarrow$ conjunto das vizinhanças $\mathcal{N}$, descritas na Seção 3.4, em ordem aleatória;
$k \leftarrow 1$;
enquanto ($k \leq r$) **faça**
  Encontre o melhor vizinho $s' \in \mathcal{RN}^{(k)}(s)$;
  se ($f(s') < f(s)$) **então**
    $s \leftarrow s'$;
    $k \leftarrow 1$;
    {Intensificação nas máquinas alteradas}
    $s \leftarrow$ *BuscaLocal_movimentoTroca*(s);
    $s \leftarrow$ *BuscaLocal_movimentoRealocaçãoMonoTarefa*(s);
    $s \leftarrow$ *BuscaLocal_movimentoRealocaçãoMultitarefa*(s) com $k = 2, 3, 4$;
    $s \leftarrow$ *BuscaLocal_movimento2-Opt*(s);
  senão
    $k \leftarrow k + 1$;
  fim se
fim enquanto
Retorne s;

3.6 Perturbações

As perturbações são realizadas por um dos dois procedimentos seguintes: Múltiplos Movimentos de Troca e Múltiplos Movimentos de Realocação Monotarefa. Os procedimentos consistem em k aplicações sucessivas dos movimentos citados, sendo k um valor inteiro aleatório entre 1 e 3.

3.7 Estrutura Auxiliar de Dados

Uma Estrutura Auxiliar de Dados, baseada no trabalho de Penna et al. (2013), foi implementada no algoritmo proposto com a intenção de armazenar informações importantes relacionadas às soluções durante o processo de busca local. Toda vez que se realiza um movimento, parte da solução é modificada; logo é necessário reavaliá-la para averiguar se houve melhora ou não. Portanto, faz-se necessário uma técnica para guiar o processo de busca e fornecer um suporte ao método de reavaliação da solução, já que um conjunto organizado de dados é de fácil acesso e muito útil para orientar o algoritmo nesta fase da execução.

Com o intuito de aperfeiçoar a busca nas vizinhanças, adotou-se uma matriz bidimensional que armazena informações importantes sobre cada uma das máquinas:

1. **Job:** Indica o índice do *job* que está em execução em cada umas das máquinas;
2. **Tarefa:** Indica o índice da Tarefa que está em execução em cada umas das máquinas;
3. **Tempo:** Indica o tempo acumulativo da Tarefa i do *Job* j, ou seja, associado a cada tarefa há o tempo total transcorrido do processo em relação às tarefas anteriores da máquina.
4. **Máquina:** Indica a máquina que está responsável por executar a Tarefa i do *Job* j;
5. **Ordem:** Indica qual a ordem que a Tarefa i do *Job* j assume durante a execução na máquina.

Esta matriz bidimensional é declarada como uma estrutura, sendo que cada linha representa uma máquina. A célula desta matriz é capaz de armazenar os cinco diferentes tipos de dados descritos acima e deve ser atualizada com os valores correntes toda vez que se realiza um movimento. Logo, os valores *Job*, *Tarefa*, *Tempo*, *Máquina* e *Ordem* para cada umas das tarefas i de um determinado *Job* j são atualizados de acordo com o processo de busca. Quando um movimento é aplicado e consequentemente a solução é alterada faz-se necessário recalcular os novos valores que serão gravados nessa matriz (estrutura auxiliar de dados). Em seguida, o método de reavaliação utiliza esses valores que servirão para orientar esta etapa do algoritmo.

Assim, depois que uma nova solução é gerada, o próximo passo é avaliá-la para verificar o quão melhor ou pior ela é. Essa avaliação torna-se mais eficaz quando, por meio das informações fornecidas pela estrutura auxiliar de dados, o algoritmo é capaz de identificar a partir de onde a solução foi modificada. Identificado este ponto, é necessário avaliar a solução apenas dessa parte em diante. Logo, poupa-se esforço computacional avaliando apenas o que fora modificado na solução e não toda ela.

Além do mais, é possível obter mais informações inerentes ao sequenciamento de máquinas simplesmente associando a estrutura auxiliar de dados com a matriz de tempo que é retirada do problema-teste. Desta maneira é facil identificar a escala de tarefas a serem executadas em uma determinada máquina, quanto tempo de processamento é necessário, quantificar a ociosidade de uma máquina e sobretudo informar as modificações na solução para orientar melhor o processo de busca.

Portanto, a estrutura auxiliar de dados não só ajuda no processo de reavaliação rápida, como também contribui para construir estatísticas referentes ao algoritmo, facilitando o trabalho de calibrá-lo e auxiliando a desenvolver novas técnicas que podem ser usadas para melhorar a qualidade das soluções, como, por exemplo, na construção da solução inicial.

4 Resultados

O algoritmo `TWTJSSP-ILS` foi codificado na linguagem de programação C++ com auxílio do *framework OptFrame* (Coelho et al., 2011). Para testá-lo, foi usado um microcomputador com processador Intel *Core 2 Duo*, 2,00 GHz e 3 GB de memória RAM e sistema operacional Windows Vista. Apesar de o microcomputador possuir dois núcleos, o algoritmo proposto não explora multiprocessamento.

Para validá-lo, foram usados 22 problemas-teste da literatura de tamanho 10×10, ou seja, com 10 máquinas e 10 *jobs*: abz5 e abz6 de Adams et al. (1988), LA16-LA24 de Lawrence (1994) e ORB01-ORB10 de Applegate e Cook (1991), que podem ser visualizados na Tabela 1. Os parâmetros adotados, obtidos experimentalmente em uma bateria preliminar de testes, foram os seguintes: *maxIter* = 10000, *time* = 120 segundos para a execução do GRASP e fator da data de entrega f = *1,3*. Esta fator é multiplicado pela data de entrega de cada *job* com a intenção de tornar o problema mais flexível, ou seja, é inserida uma margem tolerável para a conclusão do processo. Na literatura encontram-se problemas-teste que são testados com fatores superiores ao adotado neste trabalho, como por exemplo *1,5* e *1,6*. Porém, no presente trabalho, adotou-se *1,3* que caracteriza o fator que torna o problema mais difícil, uma vez que representa a menor data de entrega em relação aos demais fatores. Logo, o algoritmo é testado no maior grau de dificuldade do

problema de *Job-Shop* abordado.

Dado seu caráter estocástico, o algoritmo foi executado 50 vezes em cada problema-teste. Os resultados alcançados por `TWTJSSP-ILS` podem ser visualizados na Tabelas 1, onde o algoritmo proposto foi comparado com outros três algoritmos da literatura: o algoritmo heurístico que combina a técnica *large step random walk* de Kreipl (2000), o algoritmo genético com ILS de Essafi et al. (2008) e a Busca Tabu com *Shifting Bottleneck* de Bülbül (2011). Na Tabela 1, a coluna "Problema" indica o nome da instância e a coluna "MelhorLit" são os valores ótimos para cada um dos problemas-teste. As colunas "Melhor" e "$Desv^{Melhor}$" refere-se respectivamente, ao melhor valor encontrado por cada algoritmo e o desvio em relação ao melhor da literatura. O desvio de uma instância i é calculado pela equação (2), onde $Melhor_i^{Lit.}$ é o melhor valor da literatura para a instância i e $Melhor_i^{Alg}$ representa o melhor valor obtido pelo respectivo algoritmo.

$$Desv_i^{Melhor} = (Melhor_i^{Alg} - Melhor_i^{Lit.})/Melhor_i^{Lit.} \quad (2)$$

Tabela 1. Resultados (Instâncias: 10 × 10)

Problema	MelhorLit	*LSRW(200)* - Kreipl (2000)		*GLS* - Essafi et al. (2008)		*SB–TS* - Bülbül (2011)		*TWTJSSP-ILS*		
-	-	**Melhor**	**Desv^{Melhor}**	**Melhor**	**Desv^{Melhor}**	**Melhor**	**Desv^{Melhor}**	**Melhor**	**Desv^{Melhor}**	**Tempo (s)**
abz5	**1403**	1451	0,03	**1403**	0,00	1487	0,06	1440	0,03	240,00
abz6	**436**	**436**	0,00	**436**	0,00	**436**	0,00	**436**	0,00	367,54
la16	**1169**	1170	0,00	**1169**	0,00	**1169**	0,00	1185	0,01	410,50
la17	**899**	900	0,00	**899**	0,00	**899**	0,00	**899**	0,00	167,98
la18	**929**	**929**	0,00	**929**	0,00	**929**	0,00	960	0,03	219,64
la19	**948**	951	0,00	**948**	0,00	955	0,01	970	0,02	186,78
la20	**805**	809	0,00	**805**	0,00	**805**	0,00	**805**	0,00	203,62
la21	**463**	464	0,00	**463**	0,00	**463**	0,00	**463**	0,00	138,61
la22	**1064**	1086	0,02	**1064**	0,00	1084	0,02	1086	0,02	291,82
la23	**835**	875	0,05	**835**	0,00	877	0,05	841	0,01	248,40
la24	**835**	**835**	0,00	**835**	0,00	**835**	0,00	**835**	0,00	172,71
orb01	**2568**	2616	0,02	2570	0,00	2630	0,02	2593	0,01	568,29
orb02	**1408**	1434	0,02	**1408**	0,00	**1408**	0,00	1427	0,01	319,40
orb03	**2111**	2204	0,04	**2111**	0,00	2186	0,03	2195	0,04	653,19
orb04	**1623**	1674	0,03	**1623**	0,00	1652	0,02	1678	0,03	491,37
orb05	**1593**	1667	0,05	1662	0,04	1667	0,05	1615	0,01	528,11
orb06	**1790**	1802	0,01	**1790**	0,00	**1790**	0,00	1802	0,01	381,59
orb07	**590**	618	0,05	**590**	0,00	616	0,04	**590**	0,00	125,71
orb08	**2429**	2554	0,05	2439	0,00	2503	0,03	2503	0,03	518,56
orb09	**1316**	1334	0,01	**1316**	0,00	**1316**	0,00	1345	0,02	482,91
orb10	**1679**	1775	0,06	**1679**	0,00	1801	0,07	1753	0,04	517,71
Média	**-**	-	0,02	-	0,00	-	0,02	-	0,02	34,64

Considerando o conjunto de todos os 22 problemas-teste (Tabela 1), em termos de número de melhores resultados, tem-se: Essafi et al. (2008): 18; Bülbül (2011): 10; `TWTJSSP-ILS`: 6 e Kreipl (2000): 3. Já em termos de desvio médio, tem-se: Essafi et al. (2008): 0,00, Bülbül (2011): 0,02, Kreipl (2000): 0,02 e `TWTJSSP-ILS`: 0,02. Logo, o algoritmo proposto supera o de Kreipl (2000) em relação ao número de melhores soluções encontradas, no caso 6, contra 3, além de obter desvio semelhante aos trabalhos de Bülbül (2011) e Kreipl (2000), que juntos alcançam 0,02.

5 Conclusões

Este trabalho teve seu foco no Problema de Sequenciamento de Máquinas do tipo *Job-Shop Scheduling* com penalidade pelo tempo de atraso em relação a data de entrega. Dada a dificuldade de resolução desse problema em tempos computacionais aceitáveis, foi

proposto um algoritmo heurístico que combina os procedimentos *Iterated Local Search*, *Variable Neighborhood Descent* e uma estrutura auxiliar de dados para avaliar mais eficientemente uma solução.

O algoritmo, nomeado `TWTJSSP-ILS`, foi testado em 22 problemas-teste da literatura e comparado com três algoritmos eficientes da literatura. Claramente, o algoritmo de Essafi et al. (2008) foi o de melhor desempenho em todos os quesitos analisados. Entretanto, esse algoritmo possui alta complexidade de implementação. Esses últimos, por sua vez, têm desempenho bastante semelhante, com destaque para o `TWTJSSP-ILS` pelo fato de produzir soluções com o segundo menor desvio médio juntamente com Bülbül (2011) e Kreipl (2000) e se aproximarem muito aos melhores resultados encontrados na literatura. Além disso, o `TWTJSSP-ILS` é um algoritmo mais simples de ser implementado e requer a calibragem de muito menos parâmetros.

É importante salientar a utilidade da estrutura auxiliar de dados. De fato, por meio dessa estrutura o procedimento de reavaliação da solução torna-se rápido e eficaz, uma vez que no processo de reavaliação da solução apenas a parte alterada da solução é reavaliada, poupando considerável esforço computacional. O uso dessa estrutura é uma contribuição do presente trabalho, dado que não há registro de seu uso em problemas como este.

Muitos aperfeiçoamentos ainda podem ser feitos no algoritmo proposto com o intuito de melhorar a qualidade de seus resultados. Como trabalho futuro pretende-se implementar um módulo de Busca Tabu para servir de busca local junto ao VND, aplicar uma técnica usando o método exato *Branch and Bound* para resolver o problema individual em cada uma das máquinas, implementar mais estruturas de vizinhança para explorar melhor o espaço de soluções, além de testar o algoritmo proposto em instâncias mais complexas disponíveis na literatura.

Agradecimentos

Os autores agradecem à Universidade Federal de Ouro Preto (UFOP), ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) referente aos processos 202381/2012-9 (Programa Ciência sem Fronteiras) e 114629/2013-7 (PIBIC-UFOP), à Fundação de Amparo à Pesquisa do estado de Minas Gerais (FAPEMIG) e à Universitat Pompeu Fabra pelo apoio ao desenvolvimento deste trabalho.

Referências

Adams, J.; Balas, E. e Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, v. 34, p. 391–401.

Applegate, D. e Cook, W. (1991). A computational study of the job shop scheduling problem. *Journal of Computing*, v. 3, p. 149–156.

Bülbül, K. (2011). A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem. *Computers & Operations Research*, v. 38, p. 967–983.

Coelho, I. M.; Munhoz, P. L. A.; Haddad, M. N.; Coelho, V. N.; Silva, M. M.; Souza, M. J. F. e Ochi, L. S. (2011). A computational framework for combinatorial optimization problems. *VII ALIO/EURO Workshop on Applied Combinatorial Optimization*, v. 1, p. 51–54.

Essafi, I.; Mati, Y. e Pérès, S. D. (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, v. 35, p. 2599–2616.

Feo, T. A. e Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133.

Gonçalves, J. F.; Mendes, J. J. M. e Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, v. 167, p. 77–95.

Jain, A. S. e Meeran, S. (1999). Deterministic job-shop scheduling: past, present and future. *European Journal of Operational Research*, v. 113, p. 390–434.

Kreipl, S. (2000). A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, v. 3, p. 125–138.

Laarhoven, P. J. M. Van; Aarts, E. H. L. e Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, v. 40, p. 113–125.

Lawrence, S. *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques*. PhD thesis, Carnegie Mellon University, Estados Unidos, (1994).

Lourenço, H. R. (1995). Job-shop scheduling: computational study of local search and large-step optimization methods. *European Journal of Operational Research*, v. 83, p. 347–364.

Mine, Marcio T.; Silva, M. S. A.; Ochi, L. S. e Souza, M. J. F. (2010). O problema de roteamento de veículos com coleta e entrega simultânea: uma abordagem via iterated local search e genius. *Transporte em transformação XIV: trabalhos vencedores do prêmio CNT de Produção Acadêmica 2009*, p. 59–78. Editora Positiva, Brasília.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, v. 24, p. 1097–1100.

Nowicki, E. e Smutnicki, C. (1996). A fast tabu search algorithm for the job shop problem. *Management Science*, v. 42, p. 797–813.

Penna, P. H. V.; Subramanian, A. e Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, v. 19, p. 201–232.

Pinedo, M. e Singer, M. (1998). A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, v. 30, p. 109–118.

Pinedo, M. e Singer, M. (1999). A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, v. 46, p. 1–17.

Volta, R.; Croce, G. Della e Tadei, F. (1995). A genetic algorithm for the job shop scheduling problem. *Computers & Operations Research*, v. 22, p. 15–24.

Walter, C. *Planejamento e Controle da Produção - PCP*. Tese de doutorado, Universidade Federal do Rio Grande do Sul, Porto Alegre, (1999).