LI Simpósio Brasileiro de Pesquisa Operacional
Limeira–SP, 02 a 06 de setembro de 2019

UM ALGORITMO *SMART ITERATED LOCAL SEARCH* PARA MINIMIZAR O *MAKESPAN* E O CONSUMO TOTAL DE ENERGIA NO PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS IDÊNTICAS

Klevison Daniel de Oliveira Ribeiro[1], Luciano Perdigão Cota[1], Helen Costa[2], Frederico Gadelha Guimarães[3], Marcone Jamilson Freitas Souza[1]

[1]Programa de Pós-Graduação em Inst., Controle e Automação de Processos de Mineração, Universidade Federal de Ouro Preto e Instituto Tecnológico Vale, CEP 35.400-000, Ouro Preto (MG), Brasil

[2]Departamento de Computação e Sistemas – Universidade Federal de Ouro Preto (UFOP), CEP 35931-008, João Monlevade (MG), Brasil

[3]Departamento de Engenharia Elétrica – Universidade Federal de Minas Gerais (UFMG), CEP 31.270-901, Belo Horizonte (MG), Brasil

klevisonribeiro@aluno.ufop.edu.br, luciano.p.cota@itv.org, helen@ufop.edu.br, fredericoguimaraes@ufmg.br, marcone@ufop.edu.br

RESUMO

Este trabalho trata um problema de sequenciamento de tarefas em máquinas idênticas. No problema abordado, tem-se um conjunto de tarefas que devem ser alocadas a um conjunto de máquinas idênticas em um determinado instante de tempo, tendo como objetivo minimizar a soma ponderada do *makespan* e do consumo total de energia envolvido nessa operação. Problemas desta classe, conhecidos como sequenciamento verde, têm surgido na literatura dada a crescente preocupação com o meio ambiente e o desenvolvimento sustentável. Para resolvê-lo foram desenvolvidos dois algoritmos baseados na metaheurística *Iterated Local Search* (ILS), um tradicional e outro adaptado, nomeado *Smart* ILS, ambos tendo o *Randomized Variable Neighborhood Descent* como método de busca local. Os resultados dos algoritmos propostos foram comparados entre si e validados pelo CPLEX. Pelos experimentos computacionais, observou-se que os dois algoritmos heurísticos são capazes de produzir soluções de boa qualidade em baixo tempo de processamento, com superioridade do *Smart* ILS.

PALAVRAS CHAVE. Sequenciamento verde, Sequenciamento em Máquinas Paralelas Idênticas, Busca Local Iterada.
Tópicos: PO na Indústria.

ABSTRACT

This work deals with an identical machine scheduling problem. In this problem, we have a set of jobs that must be allocated to a set of identical machines at a given instant of time, aiming at minimizing the weighted sum of the makespan and the total energy consumption (TEC) involved in this operation. Problems of this class, known as green scheduling, have been emerging in the literature given the growing concern about the environment and sustainable development. In order to solve this problem, two algorithms based on the Iterated Local Search (ILS) metaheuristic were developed, having as local search the Randomized Variable Neighborhood Descent (RVND). The results of the proposed algorithms were compared to each other and validated by CPLEX. From the computational experiments, it was observed that the two heuristic algorithms are able to produce good quality solutions in short processing time, with superiority of the Smart ILS algorithm.

KEYWORDS. Green scheduling, Identical Machine Scheduling, Iterated Local Search.
Paper topics: OR in Industry.

https://proceedings.science/p/106872?lang=pt-br

1. Introdução

Este trabalho trata um problema de sequenciamento em máquinas idênticas (PSMI) por meio de uma abordagem que surgiu recentemente na área de sequenciamento de tarefas, o chamado sequenciamento verde (*green scheduling*) [Bampis et al., 2015; Mansouri et al., 2016; Mansouri, 2016]. Tal como em Wang et al. [2018], procura-se alocar um dado número de tarefas a um conjunto de máquinas disponíveis em determinado instante de tempo de forma a minimizar o tempo total de processamento (*makespan* ou C_{max}) e o consumo total de energia (TEC). No entanto, ao contrário de Wang et al. [2018], que tratou o problema de forma multiobjetivo, neste trabalho considera-se uma função objetivo mono-objetivo, dada pela soma ponderada dos dois objetivos considerados.

O tema *green scheduling* vem sendo amplamente estudado atualmente dada a crescente preocupação com o meio ambiente e o desenvolvimento sustentável, assim como pelas inúmeras aplicações na indústria de manufatura e automobilística [Wang et al., 2018]. Nessa versão de sequenciamento, além da busca pela minimização dos custos relacionados ao tempo de processamento (*makespan*, tempo total de atraso, entre outros), procura-se também minimizar o consumo de recursos, como água, energia elétrica ou mesmo a redução da emissão de poluentes, uma vez que todos esses fatores impactam diretamente o meio ambiente.

Trabalhos desta natureza têm marcado forte presença na literatura atualmente. Wang et al. [2018] tratam o Problema de Sequenciamento em Máquinas Idênticas objetivando a minimização do *makespan* e do TEC em uma abordagem multiobjetivo. O consumo de energia é calculado com base nos tempos de uso das máquinas e no preço de utilização delas no período de uso. Para resolvê-lo, os autores aplicaram os métodos ϵ-restrito aumentado e NSGA-II.

Cota et al. [2018] trataram o Problema de Sequenciamento em Máquinas Paralelas não relacionadas com tempos de preparação dependentes da sequência. Assim como neste trabalho, os autores buscam minimizar também o *makespan* e o TEC. Porém, o cálculo do TEC difere daquele considerado em Wang et al. [2018]. Como mencionado anteriormente, o TEC em Wang et al. [2018] é calculado com base nos tempos de uso das máquinas e no preço de utilização delas nos períodos de uso. Já em Cota et al. [2018], o TEC é calculado com base nos modos de operação das máquinas, que podem ter sua velocidade aumentada para reduzir o tempo de processamento das tarefas; entretanto, consumindo mais energia nesse processo.

Em Li et al. [2011] o PSMI é tratado com o objetivo de se reduzir o *makespan*, restringido pelo consumo limitado de recursos. Para resolvê-lo é aplicado um algoritmo baseado em *Simulated Annealing*. Ji et al. [2013], por sua vez, utilizaram um algoritmo baseado em *Particle Swarm Optimization* (PSO) para resolver o Problema de Sequenciamento em Máquinas Paralelas Uniformes. Neste caso, os autores resolvem o problema considerando um limite superior de *makespan* predefinido, e buscam a minimização do consumo de energia elétrica, água e emissão de gás carbônico.

Já em Ding et al. [2016], o Problema de Sequenciamento em Máquinas Paralelas é tratado para minimizar o custo total de eletricidade com tempo limite de execução. Os autores propõem uma abordagem exata baseada em programação linear inteira mista e também um geração de colunas para resolvê-lo.

Um modelo biobjetivo para o problema de sequenciamento em máquinas paralelas é proposto em Zeng et al. [2018]. Os autores buscam minimizar o custo total da energia elétrica utilizada bem como o número de máquinas usadas durante os horários com tarifação mais cara. Em Wu e Che [2019] também é usado um modelo biobjetivo para sequenciamento em máquinas. Entretanto, os autores tratam o problema em máquinas paralelas não relacionadas, e desenvolvem um algoritmo memético de evolução diferencial (MDE), com o objetivo de minimizar o *makespan* e o TEC.

Neste trabalho trata-se o modelo de Wang et al. [2018] em uma abordagem mono-objetivo,

em que os objetivos de minimização do *makespan* e do TEC são ponderados. Dado o fato de que o problema aqui tratado tem como subproblema o PSMI com o objetivo de minimizar o *makespan*, e este é NP-difícil [Gary e Johnson, 1979; Karp, 1972], para resolvê-lo de maneira eficiente foram desenvolvidos dois algoritmos baseados na metaheurística *Iterated Local Search* - ILS [Lourenço et al., 2003], sendo que um deles segue o *framework* tradicional e o outro introduz uma modificação no método, aqui nomeado (*Smart ILS*), seguindo a nomemclatura de Reinsma et al. [2018]. O ILS foi escolhido tendo em vista sua aplicação bem sucedida na resolução de vários outros problemas combinatórios Lourenço et al. [2003]. Os resultados desses algoritmos são comparados entre si e também com os produzidos pela formulação de programação matemática de Wang et al. [2018], adaptada para considerar a soma ponderada dos objetivos.

O restante deste trabalho está estruturado como segue. Na Seção 2 as particularidades e restrições do problema são descritas. Na Seção 3 são apresentados a representação adotada para a solução do problema, o método de geração da solução inicial, a função de avaliação adotada, as estruturas de vizinhança utilizadas para explorar o espaço de soluções do problema e os algoritmos baseados no ILS desenvolvidos para resolver o problema. Na Seção 4 são reportados os resultados dos experimentos realizados. Por fim, na Seção 5 são apresentadas as conclusões sobre a aplicação do algoritmo proposto e apresentadas sugestões para trabalhos futuros.

2. CARACTERIZAÇÃO DO PROBLEMA

O Problema de Sequenciamento em Máquinas Idênticas (PSMI) considerado neste trabalho tem as seguintes características:

1. Tem-se um conjunto de n tarefas a serem executadas em um conjunto de m máquinas durante um horizonte $T_{\max}$ de processamento;
2. As máquinas são idênticas, dessa forma o tempo de processamento de cada tarefa é sempre o mesmo, qualquer que seja a máquina usada para processá-la;
3. Cada máquina j tem uma taxa de consumo de energia, dada por e_j;
4. A cada instante k do horizonte de planejamento há um custo c_k para processar uma tarefa;
5. Uma vez iniciada uma tarefa, ela deve ser concluída, não podendo sua execução ser interrompida e retomada posteriormente;
6. Cada tarefa deve ser executada uma única vez;
7. O objetivo é minimizar a soma ponderada do *makespan* e do TEC, normalizados, dada pela expressão $\alpha \cdot C'_{\max} + \beta \cdot TEC'$, sendo α e β pesos de ponderação tais que $\alpha + \beta \leq 1$, $\alpha, \beta \in [0, 1]$, e $C'_{\max}$ e TEC' representam os valores de $C_{\max}$ e TEC normalizados no intervalo [0,1], respectivamente.

De acordo com a notação de Pinedo [2008], o PSMI aqui descrito pode ser representado pela tripla: $P_m \mid\mid (C_{\max}, TEC)$, em que P_m representa a característica do problema de utilizar máquinas paralelas idênticas e o par (C_{max}, TEC) indica os objetivos a serem minimizados.

3. DESCRIÇÃO DO ALGORITMO PROPOSTO

Nesta seção os algoritmos propostos são apresentados. Na Seção 3.1 mostra-se como uma solução do problema é representada. Na Seção 3.2 descreve-se como uma solução inicial é gerada, enquanto na Seção 3.3 mostra-se como uma solução é avaliada. Na Seção 3.4 são apresentados os movimentos utilizados para explorar o espaço de soluções do problema, enquanto na Seção 3.5 são detalhados o ILS e o Smart ILS.

3.1. Representação da Solução

Uma solução s do problema é representada tal como em Wang et al. [2018], isto é, por um vetor s de dimensão $2n$. Nesse vetor, cada posição j, com $j \in [1, n]$, indica a máquina que executa a tarefa j, enquanto que a posição $k = j + n$, com $k \in [n + 1, 2n]$ indica o tempo de início dessa tarefa.

Além disso, uma representação auxiliar foi utilizada para facilitar a implementação. Mais precisamente, a cada máquina associa-se um vetor com $T_{\max}$ posições, sendo que cada posição desse vetor representa um instante da máquina no horizonte de planejamento e cada célula desse vetor indica a tarefa que está sendo executada. No caso de a máquina estar ociosa naquele instante, atribui-se o número 0 àquela célula. A Figura 1 ilustra um exemplo de como uma solução é representada em um problema com 6 tarefas e 3 máquinas. Nessa figura percebe-se, por exemplo, que na solução s a tarefa 6 (posição 6 do vetor s) é executada na máquina 2 (valor da célula s_6 do vetor s) e no instante 5 (valor da célula s_{12} do vetor s). Na estrutura auxiliar mostra-se um vetor de 10 posições para cada máquina, já que neste exemplo $C_{\max} = 10$. Assim, na máquina M1, por exemplo, em sua quinta posição a célula contém o valor 5, indicado que no instante 5 essa máquina está processando a tarefa 5. Nesse mesmo vetor da máquina M1 tem-se o conteúdo 0 nas posições 8, 9 e 10, indicando que nesses instantes a máquina está ociosa.

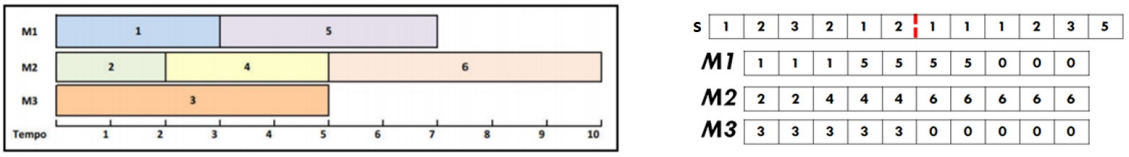


Figura 1: Representação de uma solução do problema

3.2. Solução Inicial

Uma solução inicial é gerada por um procedimento guloso, conforme descrito a seguir. As n tarefas são alocadas uma a uma, iniciando-se pela tarefa 1, à máquina que tenha o menor tempo total de processamento. Em caso de empate, a tarefa é alocada à máquina com o menor índice.

A Figura 1 ilustra esse processo de construção. Nesse exemplo existem 3 máquinas e 6 tarefas, sendo os tempos de processamento iguais a 3, 2, 5, 3, 4 e 5 unidades de tempo respectivamente. Inicialmente, as três máquinas têm tempo total de processamento igual a 0. Assim, a tarefa 1 é alocada à máquina 1, a segunda tarefa à máquina 2 e a tarefa 3 à máquina 3. No momento de alocar a tarefa 4, a máquina 1 está sendo utilizada durante 3 unidades de tempo, enquanto as máquinas 2 e 3 estão ocupadas durante 2 e 5 unidades de tempo, respectivamente. Dessa forma, a tarefa 4 será alocada à máquina 2, que tem o menor tempo total de processamento no instante de sua alocação.

A tarefa 5 é a próxima a ser alocada e, nesse momento, as máquinas 1, 2 e 3 estão sendo utilizadas até os tempos 3, 5 e 5, respectivamente. Sendo assim, a máquina 1 tem o menor tempo de processamento e ela será utilizada para executar a tarefa 5. Por fim, no instante em que a tarefa 6 deve ser alocada, as máquinas 2 e 3 estarão com o mesmo tempo total de processamento, isto é, 5 unidades de tempo, enquanto a máquina 1 estará ocupada por 7 unidades de tempo. Então a tarefa 6 é alocada à máquina 2, que é a máquina com menor tempo de ocupação e com o menor índice.

3.3. Função de Avaliação

Uma solução s do problema é avaliada pela função mono-objetivo $f(s)$, dada pela Eq. (1), que é a soma ponderada do *makespan* e do consumo total de energia, ambos normalizados no intervalo [0,1].

https://proceedings.science/p/106872?lang=pt-br

$$f(s) = \alpha \cdot C'_{\max}(s) + \beta \cdot TEC'(s) \quad (1)$$

em que $C'_{\max}$ e TEC' representam os valores de $C_{\max}$ e TEC normalizados no intervalo [0,1], e α e β, com $\alpha, \beta \in [0, 1]$, $\alpha + \beta \leq 1$, são pesos de ponderação que refletem a importância relativa de cada grandeza avaliada.

A determinação de $C'_{\max}(s)$ na Equação (1) é obtida de acordo com a Equação (2):

$$C'_{\max}(s) = C_{\max}(s)/T_{\max} \quad (2)$$

sendo $C_{\max}(s)$ o maior tempo total de processamento dentre todas as máquinas e $T_{\max}$ o horizonte de planejamento, isto é, um limite superior para $C_{\max}$.

Para estimar $T_{\max}$ basta minimizar apenas a parcela $TEC'(s)$ da solução. Isto é possível pois as duas parcelas que compõem a função de avaliação são conflitantes. Sendo assim, ao se minimizar uma, a outra atinge o seu maior valor possível. Então, para estimar o valor de $T_{\max}$, basta minimizar a parcela $TEC'(s)$ alocando as tarefas nos instantes em que o preço de utilização de energia for mais baixo. Isso criará lacunas no horizonte de planejamento, pois os períodos de tempo com preço alto de utilização de energia não terão tarefas alocadas. Assim, a parcela $C'_{\max}(s)$ alcançará o seu valor máximo, isto é, uma estimativa do limite superior $T_{\max}$ para o *makespan*.

Por sua vez, a determinação de $TEC'(s)$ na Equação (1) é feita de acordo com a Equação (3):

$$TEC'(s) = TEC(s)/TEC_{\max} \quad (3)$$

em que $TEC(s)$ mensura o consumo total de energia de todas as máquinas envolvidas na solução s e é obtida de acordo com a Equação (4):

$$TEC(s) = \sum_{j=1}^{n} EC_j(s) \quad (4)$$

sendo EC_j o consumo de energia na máquina j, calculado por meio da Equação (5):

$$EC_j(s) = e_j \cdot \sum_{i=1}^{n} \sum_{k=1}^{T_{\max}} c_k x_{ijk} \quad (5)$$

Na Equação (5), e_j é a taxa de consumo de energia da máquina j, c_k representa o custo de uso da energia no instante de tempo k, $T_{\max}$ é o horizonte de planejamento e, portanto, o limite superior para $C_{\max}$.

Na Equação (3), $TEC_{\max}$ representa o consumo máximo de energia e pode ser estimado alocando-se todas as tarefas na máquina de maior taxa de consumo de energia, propiciando um total de consumo de energia grande o suficiente para normalizar a parcela $TEC(s)$ no intervalo [0,1].

A Figura 1 ilustra a representação de uma solução considerada para mostrar o cálculo das parcelas $C_{\max}(s)$ e $TEC(s)$. Os tempos de processamento das máquinas $M1$, $M2$ e $M3$ nessa solução são iguais a $C_{M1} = 7$, $C_{M2} = 10$ e $C_{M3} = 5$, respectivamente. Como $C_{\max}(s)$ é o maior valor de C_j dentre todas as máquinas, então $C_{\max}(s) = 10$.

A energia consumida por cada máquina j, por sua vez, é calculada com base na taxa de consumo de cada máquina e no tempo de uso de cada máquina. Dessa forma, considerando as taxas de consumo das máquinas 1, 2 e 3, respectivamente iguais a 1, 3 e 1 unidades e os custos de uso nos instantes de tempo de 1 a 10 iguais a 6, 6, 5, 5, 5, 2, 2, 2, 2, 2 respectivamente, temos que:

$$TEC(s) = EC_1(s) + EC_2(s) + Ec_3(s) = 31 + 111 + 27 = 169$$

É importante ressaltar que quando movimentos são realizados para analisar a vizinhança da solução corrente, é necessário atualizar o valor da função de avaliação dessa solução. Para isso, um cálculo inteligente é utilizado para evitar que todas as máquinas tenham seu consumo de energia recalculados. Considerando que o movimento realizado envolve apenas as máquinas j_1 e j_2, a atualização da parcela $TEC(s)$ da função $f(s)$ obedece à seguinte regra:

$$TEC(s) = TEC^a(s) - (EC^a_{j_1} + EC^a_{j_2}) + (EC^d_{j_1} + EC^d_{j_2}) \quad (6)$$

em que $EC^a_{j_1}$ e $EC^a_{j_2}$ representam, respectivamente, o consumo de energia nas máquinas j_1 e j_2 antes do movimento e $EC^d_{j_1}$ e $EC^d_{j_2}$ representam o consumo de energia nessas mesmas máquinas depois do movimento. Nessa Equação, $EC^a(s)$ representa o consumo de energia da solução s antes da aplicação do movimento.

3.4. Estruturas de vizinhança

Foram utilizadas três estruturas de vizinhança (N_1, N_2, N_3) para explorar o espaço de soluções deste problema. Cada uma delas é gerada a partir dos seguintes movimentos, nesta ordem:

N_1) **Troca de tarefas entre máquinas:** Duas tarefas de máquinas diferentes são trocadas de posição.

N_2) **Realocação de tarefa para outra máquina:** Uma tarefa de uma máquina é realocada para outra posição de outra máquina.

N_3) **Realocação de tarefa na mesma máquina:** Uma tarefa de uma máquina é realocada para outra posição na mesma máquina, isto é, ela passa a iniciar em um instante de tempo diferente. Tal movimento é importante pois permite a redução do TEC sem alterar o $C_{\max}$.

Em todas as estruturas de vizinhança existe um rearranjo de tarefas nas máquinas. O rearranjo ocorre sempre que a tarefa realocada é inserida numa posição que já tenha alguma tarefa sendo executada. Neste caso, a tarefa sobreposta e as tarefas subsequentes são remanejadas para o término da tarefa que foi realocada primeiramente, causando a sobreposição. Entretanto, o rearranjo de tarefas não ocorre quando surgem lacunas no horizonte de planejamento das máquinas. Nesses casos, o rearranjo não ocorre porque essas lacunas podem gerar soluções melhores que a solução anterior ao movimento.

3.5. Algoritmos Propostos

Os dois algoritmos desenvolvidos para resolver o PSMI são baseados na metaheurística ILS. O primeiro algoritmo foi implementado baseado no ILS tradicional, chamado apenas de ILS. Já o segundo, denominado *Smart* ILS, foi implementado com uma adaptação em relação ao procedimento de perturbação.

3.5.1. *Iterated Local Search* (ILS)

O algoritmo ILS tem seu procedimento descrito pelo Algoritmo 1. Inicialmente, o ILS recebe duas entradas, uma solução inicial ($s_{inicial}$) e um limite máximo de iterações sem melhora ($ILS_{\max}$). O número de iterações sem melhora é o número de vezes consecutivas no qual a solução s não é atualizada. Por fim, a saída do ILS é uma solução denotada por s.

Na linha 1 do Algoritmo 1, a solução inicial $s_{inicial}$ é submetida a um procedimento de busca local pelo RVND, descrito pelo Algoritmo 3. Na linha 2, o número de iterações sem melhora é inicializado em 0 e na linha 3 o nível de perturbação é iniciada em 1. Na linha 4 inicia-se o laço

https://proceedings.science/p/106872?lang=pt-br

Algoritmo 1: ILS

```
   entrada: s_inicial, ILS_max
   saída  : s
1  s ← RVND(s_inicial);
2  iterSemMelhora ← 0;
3  nivel ← 1;
4  enquanto (iterSemMelhora ≤ ILS_max) faça
5      iterSemMelhora ← iterSemMelhora + 1;
6      s' ← perturbacao(s, nivel);
7      s'' ← RVND(s');
8      se (f(s'') < f(s)) então
9          s ← s'';
10         iterSemMelhora ← 0;
11         nivel ← 1;
12     fim
13     senão
14         nivel ← nivel + 1;
15     fim
16 fim
17 retorne s;
```

de repetição do algoritmo ILS, que é encerrado quando o número de iterações sem melhora for maior que o limite máximo $ILS_{\max}$, recebido como entrada. O número de iterações sem melhora é incrementado na linha 5. Na linha 6 é aplicada uma perturbação na solução corrente de acordo com a Seção 3.5.3, gerando-se uma solução intermediária s'.

Uma nova busca local é aplicada sobre essa solução intermediária s', gerando-se possivelmente uma solução ótima local s''. Essa etapa ocorre na linha 7. Das linhas 8 a 12 tem-se um laço condicional, no qual as soluções s'' e s são avaliadas de acordo com a Equação (1). Se s'' for melhor do que s, então a solução s é atualizada, o número de iterações sem melhora retorna para seu valor inicial, isto é, 0, e o nível de perturbação é reiniciado para seu valor mínimo, isto é, 1.

Caso s'' não represente uma melhora em relação à solução corrente s, o algoritmo executa então o laço das linhas 13 a 15, incrementando o nível de perturbação em uma unidade.

3.5.2. *Smart Iterated Local Search (Smart ILS)*

O *Smart* ILS, dado pelo Algoritmo Algoritmo 2, tem apenas uma diferença em relação ao ILS. Nele, adiciona-se o procedimento que pode ser visualizado entre as linhas 15 e 23 do Algoritmo 2. Nesse trecho do algoritmo é realizado um controle de nível da perturbação. Há um parâmetro $tentativas_{\max}$ que define quantas tentativas no mesmo nível de perturbação são permitidas para que se encontre uma solução melhor, antes de aumentar o nível da perturbação.

A implementação dessa funcionalidade se justifica pelo fato de que aumentar o nível de perturbação após uma única tentativa pode ser uma decisão precipitada. De fato, como a perturbação ocorre de forma aleatória, o vizinho gerado em um determinado nível de perturbação pode ter sido ruim para a continuação da busca, no sentido de que o procedimento de busca local aplicado à solução intermediária poderia conduzir a um único vale da região pesquisada, que não representaria uma solução de melhora. Por outro lado, se outro vizinho fosse gerado, pode ser outros vales fossem

encontrados, justificando a estratégia de somente aumentar o nível de perturbação após algumas tentativas frustradas de melhoria.

Algoritmo 2: Smart ILS

```
   entrada: s_inicial, ILS_max, tentativas_max
   saída  : s
1  s ← RVND(s_inicial);
2  iterSemMelhora ← 0;
3  nivel ← 1;
4  tentativas ← 1;
5  enquanto (iterSemMelhora ≤ ILS_Max) faça
6      iterSemMelhora ← iterSemMelhora + 1;
7      s' ← perturbacao(s, nivel);
8      s'' ← RVND(s');
9      se (f(s'') < f(s)) então
10         s ← s'';
11         iterSemMelhora ← 0;
12         nivel ← 1;
13         tentativas ← 1;
14     fim
15     senão
16         se (tentativas >= tentativas_Max) então
17             nivel ← nivel + 1;
18             tentativas ← 1;
19         fim
20         senão
21             tentativas ← tentativas + 1;
22         fim
23     fim
24 fim
25 retorne s;
```

3.5.3. Perturbação

A partir de uma solução s, uma solução intermediária s' (linha 5 do Algoritmo 1) é gerada por um procedimento de perturbação que funciona como segue. Sobre essa solução s são aplicadas *nivel* + 1 modificações na solução, sendo que cada modificação consiste em escolher aleatoriamente uma máquina e uma tarefa dessa máquina e, em seguida, realocar essa tarefa para o final do processamento de outra máquina, também escolhida de forma aleatória. Assim, quando *nivel* é igual a 1, são feitas duas modificações na solução corrente s. Quando *nivel* é igual a 2 são feitas três modificações na solução e assim por diante. Observa-se que sempre que há melhora na solução corrente, a variável *nivel* retorna ao seu valor mínimo.

3.5.4. Busca Local

A busca local dos algoritmos ILS e *Smart* ILS é feita pelo método *Randomized Variable Neighborhood Descent* – RVND [Souza et al., 2010; Haddad et al., 2014]. O Algoritmo 3 mostra seu pseudocódigo.

No Algoritmo 3, tanto a entrada quanto a saída são armazenadas na solução s. Na linha 1, um vetor v de três posições é criado com os números ordenados 1, 2 e 3, cada qual indicando uma das três vizinhanças relatadas na Seção 3.4. Na linha 2, esse vetor é embaralhado para permitir que as buscas locais ocorram em ordens diferentes à cada chamada do RVND. Na linha 3, o índice k usado para percorrer o vetor v é inicializado com o valor 1.

Um laço de repetição é iniciado na linha 4, para definir que o processo deve se repetir até que todas as três vizinhanças (N_1, N_2 e N_3) sejam analisadas. Na linha 5 é aplicada uma busca pelo primeiro vizinho de melhora na vizinhança corrente. Nesse método, retorna-se a primeira solução

Algoritmo 3: RVND

```
   entrada: s
   saída  : s
1  v ← [1 2 3];
2  embaralha(v);
3  k ← 1;
4  enquanto (k ≤ 3) faça
5      s' ← FirstNeigh(s, N_{v_k});
6      se (f(s') < f(s)) então
7          s ← s'; k ← 1;
8      fim
9      senão
10         k ← k + 1;
11     fim
12 fim
13 retorne s;
```

de melhora encontrada na vizinhança explorada ou, no caso de não haver melhora, a própria solução analisada, ambas denotadas por s'. Nas linhas 6 a 11 é verificado se a solução s' proveniente da busca local é melhor que a solução s corrente. Caso positivo, s é atualizada e a vizinhança corrente passa a ser a primeira, isto é, $k = 1$. Caso contrário, na linha 10 passa-se para próxima vizinhança, incrementando-se o índice k.

4. RESULTADOS

Os algoritmos ILS e *Smart ILS*, este último aqui denotado por SILS, por simplicidade, foram implementados na linguagem de programação C/C++, utilizando-se a IDE *NetBeans* 8.2. Os testes foram realizados em um computador Dell Inspirion 5558, com processador Intel Core i5-5200U, 2.2 GHz, 8 GB de memória RAM e sistema operacional Ubuntu 16.04 LTS de 64 bits. A função de avaliação testada, dada pela Equação (1), considerou $\alpha = \beta = 0.5$, isto é, tanto o *makespan* quanto o consumo total de energia tiveram o mesmo peso.

Um conjunto de 30 instâncias de pequeno porte, disponibilizado em Wang et al. [2018], foi utilizado com a finalidade de investigar a eficiência dos algoritmos ILS e SILS. Cada uma dessas instâncias foi executada 30 vezes, sendo coletados o tempo médio de execução do algoritmo, o melhor valor de função objetivo retornado pelos algoritmos e a média desses valores. Nesses testes, os valores utilizados para $T_{\max}$ foram fornecidos nas próprias instâncias, não sendo necessária a utilização do procedimento para estimá-lo, descrito na Seção 3.3.

Inicialmente, antes de executar os testes em todas as instâncias, buscou-se calibrar os parâmetros dos algoritmos ILS e SILS, com a finalidade de obter o melhor desempenho possível deles. Para tal, foi utilizado o pacote *irace* [López-Ibáñez et al., 2016]. Cada uma das instâncias utilizadas contém um conjunto predefinido de tarefas (n), máquinas (m) e faixas de preço no horizonte de planejamento ($T_{\max}$), com $n \in \{6, 10, 15, 20, 25\}$, $n \in \{3, 5, 7\}$ e $T_{\max} \in \{50, 80\}$. Para fazer a calibração foram utilizadas as instâncias 1, 9, 17, 20 e 28 de Wang et al. [2018], pois elas refletem um conjunto diverso com relação ao número de tarefas, máquinas e faixas de preço de energia.

No ILS, o único parâmetro calibrado foi o $ILS_{\max}$. Os valores testados para o parâmetro foram os seguintes: $ILS_{\max} \in \{500, 1000, 1500, 2000, 2500\}$. Ao final dos experimentos de calibração, a configuração com melhor desempenho foi: $ILS_{\max} = 1000$. Já para o *Smart* ILS foram calibrados os parâmetros $ILS_{\max}$ e $tentativas_{\max}$. Os valores utilizados na calibração foram: $ILS_{\max} \in \{500, 1000, 1500, 2000, 2500\}$ e $tentativas_{\max} \in \{2, 4, 6, 8, 10\}$. A melhor configuração para este caso foi $ILS_{\max}$ = 1500 e $tentativas_{\max}$ = 8.

Além dos testes com os algoritmos ILS e SILS, utilizou-se também o otimizador CPLEX,

https://proceedings.science/p/106872?lang=pt-br

versão 12.63, para resolver as instâncias com o objetivo de validar o desempenho dos algoritmos propostos.

A Tabela 1 mostra o resultado da comparação de desempenho entre o CPLEX e os algoritmos ILS e SILS. Nela, a coluna # indica o número da instância, a coluna n representa o número de tarefas da instância, a coluna m o número de máquinas e a coluna $T_{\max}$ a quantidade de instantes de tempo disponíveis para a execução das tarefas. Em seguida são apresentados os valores da função objetivo gerada pelo CPLEX (coluna CPLEX_{FO}) e o tempo de processamento, em segundos, requerido por esse resolvedor (coluna $\text{CPLEX}_{\text{Tempo}}$). Para cada um dos métodos heurísticos são apresentados o melhor valor da função objetivo encontrada (coluna Melhor FO), o valor médio em 30 execuções de cada instância (coluna FO Médio) e o tempo demandado, em segundos (coluna Tempo(s)). Por fim, é reportado o Desvio Percentual Relativo (DPR) do valor da solução produzida pelo SILS em relação ao ILS, tanto com relação ao melhor valor encontrado em cada instância, quanto em relação ao valor médio, ambos calculados com base na Equação (7):

$$DPR_i = \frac{f_i^{SILS} - f_i^{ILS}}{f_i^{ILS}} \cdot 100 \tag{7}$$

em que f_i^{ILS} representa o melhor valor (respectivamente valor médio) do ILS na i-ésima instância e f_i^{SILS} o melhor valor (respectivamente valor médio) do SILS nessa instância.

Observa-se que valores negativos de RPD na Tabela 1 indicam que o algoritmo SILS obteve desempenho melhor que o do ILS. Os valores em negrito destacam os resultados no qual o SILS teve desempenho igual ou superior ao ILS. Os valores acompanhados de * mostram que o valor da função objetivo (FO) do CPLEX foi atingido por um algoritmo e, por fim, os valores destacados com ** mostram que os resultados do CPLEX foram superados na instância destacada.

Tabela 1: Comparação CPLEX × ILS × SILS

Instâncias				CPLEX		SILS			ILS			RPD (%)	
#	n	m	$T_{\max}$	FO	Tempo (s)	Melhor FO	FO Médio	Tempo (s)	Melhor FO	FO Médio	Tempo (s)	Melhor (%)	Médio (%)
1	**6**	**3**	**50**	0,1775	4,28	**0,1775***	0,1775	2,21	0,1775*	0,1776	2,72	**0,00**	**-0,08**
2	**6**	**3**	**80**	0,0845	1,70	**0,0845***	0,0845	5,04	0,0845*	0,0864	4,43	**0,00**	**-2,28**
3	**6**	**5**	**50**	0,1203	1,79	**0,1203***	0,1216	2,97	0,1203*	0,1238	3,76	**0,00**	**-1,83**
4	**6**	**5**	**80**	0,0875	4,60	**0,0875***	0,0875	12,77	0,0875*	0,0875	6,65	**0,00**	**0,00**
5	**6**	**7**	**50**	0,1051	6,41	**0,1051***	0,1062	6,54	0,1051*	0,1053	5,06	**0,00**	0,81
6	**6**	**7**	**80**	0,1247	17,45	**0,1247***	0,1274	10,57	0,1247*	0,1376	8,78	**0,00**	**-7,42**
7	**10**	**3**	**50**	0,2405	29,02	**0,2405***	0,2422	7,40	0,2405*	0,2426	3,43	**0,00**	**-0,14**
8	**10**	**3**	**80**	0,1730	154,85	**0,1758**	0,1790	9,18	0,1775	0,1834	6,25	**-0,95**	**-2,37**
9	**10**	**5**	**50**	0,3144	924,14	**0,3171**	0,3270	5,12	0,3270	0,3371	3,88	**-3,03**	**-3,00**
10	**10**	**5**	**80**	0,1294	79,47	**0,1294***	0,1308	16,47	0,1294*	0,1358	7,71	**0,00**	**-3,72**
11	**10**	**7**	**50**	0,1384	11,55	**0,1384***	0,1452	9,12	0,1405	0,1490	5,17	**-1,54**	**-2,54**
12	**10**	**7**	**80**	0,1470	1567,75	**0,1470***	0,1490	21,42	0,1476	0,1611	10,68	**-0,46**	**-7,53**
13	**15**	**3**	**50**	0,4432	7200,00	**0,4478**	0,4608	7,19	0,4541	0,4725	18,43	**-1,37**	**-2,48**
14	**15**	**3**	**80**	0,2341	7200,00	**0,2361**	0,2436	12,88	0,2400	0,2521	8,01	**-2,04**	**-3,92**
15	**15**	**5**	**50**	0,3432	7200,00	**0,3789**	0,3815	8,20	0,3789	0,3826	31,49	**0,00**	**-0,27**
16	**15**	**5**	**80**	0,1679	7200,00	**0,1679***	0,1746	21,36	0,1703	0,1799	10,19	**-1,39**	**-2,91**
17	**15**	**7**	**50**	0,2103	161,36	**0,2124**	0,2289	16,65	0,2124	0,2367	7,12	**0,00**	**-3,31**
18	**15**	**7**	**80**	0,1551	7200,00	**0,1551***	0,1649	33,38	0,1557	0,1765	13,19	**-0,43**	**-6,55**
19	**20**	**3**	**50**	0,4253	7200,00	**0,4370**	0,4388	5,50	0,4370	0,4383	33,79	**0,00**	0,10
20	**20**	**3**	**80**	0,3889	7200,00	**0,3948**	0,4246	18,08	0,4283	0,4330	9,27	**-7,81**	**-1,95**
21	**20**	**5**	**50**	0,3027	7200,00	0,3159	0,3312	10,20	0,3003**	0,3446	56,07	5,22	**-3,87**
22	**20**	**5**	**80**	0,2341	7200,00	**0,2394**	0,2486	27,02	0,2454	0,2675	11,55	**-2,41**	**-7,07**
23	**20**	**7**	**50**	0,1819	7200,00	**0,1830**	0,1974	16,28	0,1835	0,1988	23,34	**-0,29**	**-0,68**
24	**20**	**7**	**80**	0,2392	7200,00	**0,2515**	0,2643	40,60	0,2578	0,2785	18,84	**-2,42**	**-5,13**
25	**25**	**3**	**50**	0,5416	7200,00	0,5497	0,5526	9,84	0,4992**	0,5533	52,29	10,12	**-0,13**
26	**25**	**3**	**80**	0,3024	7200,00	**0,3024***	0,3100	23,59	0,3040	0,3116	87,10	**-0,50**	**-0,53**
27	**25**	**5**	**50**	0,3500	7200,00	0,3686	0,3851	15,08	0,2607**	0,3842	87,79	41,39	0,24
28	**25**	**5**	**80**	0,2998	7200,00	**0,3098**	0,3247	30,87	0,3105	0,3368	28,39	**-0,22**	**-3,62**
29	**25**	**7**	**50**	0,3319	7200,00	0,3462	0,3774	17,82	0,3168**	0,3955	191,49	9,30	**-4,55**
30	**25**	**7**	**80**	0,3870	7200,00	**0,3826****	0,4209	45,68	0,4160	0,4736	19,24	**-8,02**	**-11,12**

Pela Tabela 1 percebe-se que o CPLEX foi capaz de provar, em duas horas de processa-

https://proceedings.science/p/106872?lang=pt-br

mento, a otimalidade das soluções em 13 instâncias. O algoritmo ILS foi capaz de encontrar o ótimo em 8 instâncias, enquanto o SILS alcançou todas as soluções ótimas geradas pelo CPLEX. Das 17 instâncias em que o CPLEX não encontrou a solução ótima em duas horas de processamento, tanto o ILS quanto o SILS foram melhores ou iguais ao CPLEX em 4 instâncias. Comparando-se os dois algoritmos heurísticos em termos da capacidade de gerar a melhor solução, das 30 instâncias, o SILS só não foi melhor que o ILS em 4 delas. Por outro lado, em termos de desempenho médio, o algoritmo SILS só não foi melhor que o ILS em 3 das 30 instâncias. Porém, destaca-se que o ILS foi mais rápido, uma vez que o tempo de execução do algoritmo SILS só foi menor em 11 instâncias.

Para investigar melhor a diferença de desempenho entre o ILS e SILS foi aplicado também o teste estatístico ANOVA [Montgomery, 2013]. Este teste foi realizado com 95% de grau de confiança, baseando-se nos valores médios da função objetivo gerados por cada algoritmo. Por meio deste teste foi obtido um valor de p-valor = 0,813, o que representa um forte indício de que não existe diferença estatística entre os resultados.

5. CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho tratou-se o Problema de Sequenciamento de Tarefas em Máquinas Idênticas com uma abordagem mono-objetivo, tendo por objetivo a minimização da soma ponderada do *makespan* e do consumo de energia (TEC). Para resolvê-lo, foram desenvolvidos dois algoritmos baseados na metaheurística ILS, tendo o RVND como método de busca local.

Os algoritmos desenvolvidos, nomeados ILS e SILS, foram testados e seus resultados comparados com aqueles produzidos pelo otimizador CPLEX aplicado à formulação adaptada de Wang et al. [2018], limitado a duas horas de processamento. Pelos experimentos, apesar de não haver diferença estatística entre os métodos, pôde ser notada uma superioridade do SILS em relação ao ILS tradicional, uma vez que os resultados do primeiro foram melhores que os do segundo em 16 das 30 instâncias no melhor caso e em 25 instâncias considerando a média dos valores.

Como trabalhos futuros, pretende-se aperfeiçoar o algoritmo SILS de forma a diminuir a variabilidade das soluções finais. Uma alternativa é fazer uma versão *multi-start* dele e outra é utilizar outras formas de perturbar as soluções ótimas locais. Além disso, propõe-se tratar o problema como sendo de otimização multiobjetivo; devendo-se, assim, retornar um conjunto de soluções não dominadas, cabendo ao tomador de decisão a escolha da solução mais apropriada para o seu interesse.

Agradecimentos

Os autores agradecem à Universidade Federal de Ouro Preto, ao Instituto Tecnológico Vale e às agências de fomento CNPq e FAPEMIG, pelo apoio ao desenvolvimento deste trabalho.

Referências

Bampis, E., Letsios, D., e Lucarelli, G. (2015). Green scheduling, flows and matchings. *Theoretical Computer Science*, 579:126–136.

Cota, L. P., Coelho, V. N., Guimarães, F. G., e Souza, M. J. F. (2018). Bi-criteria formulation for green scheduling with unrelated parallel machines with sequence-dependent setup times. *International Transactions in Operational Research*. Disponível em https://doi.org/10.1111/itor.12566.

Ding, J.-Y., Song, S., Zhang, R., Chiong, R., e Wu, C. (2016). Parallel machine scheduling under time-of-use electricity prices: New models and optimization approaches. *IEEE Transactions on Automation Science and Engineering*, 13(2):1138–1154.

Gary, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York.

Haddad, M. N., Cota, L. P., Souza, M. J. F., e Maculan, N. (2014). AIV: A heuristic algorithm based on iterated local search and variable neighborhood descent for solving the unrelated parallel machine scheduling problem with setup times. In *Proceedings of the 16th International Conference on Enterprise Information Systems (ICEIS 2014)*, p. 376–383, Lisboa, Portugal.

Ji, M., Wang, J.-Y., e Lee, W.-C. (2013). Minimizing resource consumption on uniform parallel machines with a bound on makespan. *Computers & Operations Research*, 40(12):2970–2974.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, p. 85–103. Springer.

Li, K., Shi, Y., Yang, S.-l., e Cheng, B.-y. (2011). Parallel machine scheduling problem to minimize the makespan with resource dependent processing times. *Applied Soft Computing*, 11(8):5551–5557.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., e Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

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

Mansouri, E., S Afshin e Aktas (2016). Minimizing energy consumption and makespan in a two-machine flowshop scheduling problem. *Journal of the Operational Research Society*, 67(11): 1382–1394.

Mansouri, S. A., Aktas, E., e Besikci, U. (2016). Green scheduling of a two-machine flowshop: Trade-off between makespan and energy consumption. *European Journal of Operational Research*, 248(3):772–788.

Montgomery, D. C. (2013). *Design and Analysis of Experiments*. Joh Wiley and Sons, 8th edition.

Pinedo, M. (2008). *Scheduling: theory, algorithms and Systems*. Springer Publishing Company, Incorporated, 3ª edition.

Reinsma, J. A., Penna, P. H. V., e Souza, M. J. F. (2018). Um algoritmo simples e eficiente para resolução do problema do caixeiro viajante generalizado. In *Anais do L Simpósio Brasileiro de Pesquisa Operacional*, Rio de Janeiro, RJ. SOBRAPO.

Souza, M. J., Coelho, I. M., Ribas, S., Santos, H. G., e Merschmann, L. H. d. C. (2010). A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research*, 207(2):1041–1051.

Wang, S., Wang, X., Yu, J., Ma, S., e Liu, M. (2018). Bi-objective identical parallel machine scheduling to minimize total energy consumption and makespan. *Journal of Cleaner Production*, 193:424–440.

Wu, X. e Che, A. (2019). A memetic differential evolution algorithm for energy-efficient parallel machine scheduling. *Omega*, 82:155–165.

Zeng, Y., Che, A., e Wu, X. (2018). Bi-objective scheduling on uniform parallel machines considering electricity cost. *Engineering Optimization*, 50(1):19–36.

https://proceedings.science/p/106872?lang=pt-br