SEQUENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS CONSIDERANDO DESGASTES DEPENDENTES DA SEQUÊNCIA

Vívian Ludimila Aguair Santos
Universidade Federal de Viçosa
Campus Universitário, 36570-900, Viçosa-MG, Brazil
vivian.santos@ufv.br

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

RESUMO

Este trabalho estuda um problema de sequenciamento de tarefas em máquinas paralelas considerando que as tarefas causam certos desgastes das máquinas. Estes desgastes irão diminuir o desempenho das máquinas aumentando os tempos de processamento das tarefas ao longo do tempo. Sendo assim, busca-se o sequenciamento que diminua o desgaste das máquinas e como consequência reduza o tempo de conclusão das tarefas. Como se trata de um problema NP-difícil, para determinar soluções aproximadas de alta qualidade, foram desenvolvidos dois algoritmos heurísticos. O primeiro é baseado na meta-heurística *Iterated Local Search* (ILS) e o segundo é uma combinação do ILS com a heurística *Randon Variable Neighborhood Descent* (RVND). De acordo com o que se conhece sobre o estado da arte do problema, esta é a primeira aplicação da meta-heurística ILS para o problema abordado. Os dois algoritmos propostos são comparados com um algoritmo apresentado na literatura. Resultados e análises estatísticas são apresentados mostrando o bom desempenho dos algoritmos propostos.

PALAVRAS CHAVE. ***Scheduling*****, Máquinas Paralelas, Meta-heurísticas.**

Meta-heurísticas, Otimização Combinatória

ABSTRACT

This paper studies a job scheduling problem on parallel machines where the jobs cause certain deterioration of the machines. Such deterioration will reduce the performance of machines by increasing the processing times of the jobs over time. So, we seek a sequencing which reduces the deterioration of machines and consequently reduce the completion time of jobs. Since the problem is NP-Hard, two heuristic algorithms are proposed to obtain near-optimal solutions. The first algorithm is based on the Iterated Local Search (ILS) meta-heuristic and the second is a combination of ILS with Randon Variable Neighborhood Descent (RVND) heuristic. To the best of our knowledge, this is the first application of ILS meta-heuristic to solve the addressed problem. The two proposed algorithms are compared with an algorithm presented in the literature. Results and statistical analyzes are presented showing the good performance of the proposed algorithms.

KEYWORDS. Scheduling, Parallel Machines, Meta-heuristic.

Meta-heuristic, Combinatorial Optimization

1. Introdução

Em problemas determinísticos de *scheduling* os tempos de processamento das tarefas são conhecidos e são valores fixos, no entanto existem situações em que os tempos de processamento das tarefas podem incrementar de acordo com a ordem do processamento nas máquinas (Joo e Kim, 2015). Esse incremento é devido ao desgaste das máquinas com o passar do tempo, ou seja, a medida que as máquinas realizam as tarefas, elas sofrem degastes (Lee et al., 2010).

Os primeiros autores que consideraram o conceito de desgaste ou deterioração em problemas de programação da produção foram Gupta et al. (1987), Browne e Yechiali (1990). Os problemas de *scheduling* que consideram desgastes têm sido recentemente estudados devido suas aplicações no gerenciamento logístico nas indústrias, uma vez que o desgaste das máquinas afeta o tempo final da produção.

Na literatura existem alguns trabalhos que consideram que o tempo de processamento real de uma tarefa é dado pelo tempo de processamento desta tarefa multiplicado por um fator de deterioração da máquina. Ji e Cheng (2008), Ji e Cheng (2009), Huang e Wang (2011) consideram o problema de programação da produção em máquinas paralelas em que o tempo de processamento de uma tarefa é uma função linear crescente de seu tempo de início. Para minimizar o tempo total de conclusão das tarefas (fluxo total), Ji e Cheng (2008) desenvolvem um algoritmo de aproximação de tempo polinomial. Ji e Cheng (2009) buscam minimizar o *makespan*, a carga total da máquina e o tempo total de conclusão usando algoritmos heurísticos. Enquanto que Huang e Wang (2011) desenvolvem algoritmos heurísticos para a minimização de dois objetivos separadamente: o total de diferenças absolutas em tempos de conclusão e as diferenças totais absolutas em tempos de espera. Cheng et al. (2009) tratam do problema de sequenciamento de tarefas em uma máquina simples com efeitos de aprendizagem. Para minimizar o *makespan* e o tempo total de conclusão das tarefas são propostos algoritmos heurísticos.

Na literatura existem outros trabalhos que levam em conta a posição das tarefas no sequenciamento para determinar o tempo de processamento. Yang (2011), Yang et al. (2012) e Ma et al. (2015) definem o problema de sequenciamento em máquinas paralelas considerando simultaneamente os efeitos de desgastes dependente das posições das tarefas e atividades de manutenção. Ji et al. (2013) consideram apenas uma máquina e propõem um algoritmo de aproximação de tempo polinomial para minimizar o *makespan*. O objetivo dos trabalhos de Yang (2011) e Yang et al. (2012) é encontrar as posições ótimas das atividades de manutenção e as sequências das tarefas de forma a minimizar a carga total da máquina. Nesses trabalhos são propostos algoritmos heurísticos. Já Ma et al. (2015) buscam minimizar o tempo total de conclusão das tarefas. Mosheiov (2012) definem uma função geral não decrescente dependente da posição das tarefas e dos desgastes das máquinas, para o sequenciamento de tarefas em máquinas paralelas, com o objetivo de minimizar a carga total. Toksarı e Güner (2009) consideram o sequenciamento em máquinas paralelas com diferentes penalidades pelo adiantamento e atraso nas entregas das tarefas. Além disso, são considerados simultaneamente os efeitos de aprendizagem baseada na posição das tarefas e o desgaste linear e não-linear. Para resolução do problema são desenvolvidos um modelo matemático e um limite inferior. Ruiz-Torres et al. (2013) consideram o sequenciamento de tarefas em máquinas paralelas onde o desgaste das máquinas são dependentes da sequência de processamento das tarefas. Para minimizar o *makespan*, os autores desenvolvem um algoritmo heurístico denominado SA* que é baseado na meta-heurística *Simulated Annealing*.

Vale ressaltar que o problema tratado neste trabalho é equivalente ao problema estudado por Ruiz-Torres et al. (2013). Neste artigo propõe-se dois algoritmos heurísticos: o primeiro é baseado na meta-heurística ILS (Lourenço et al., 2003) e o segundo na combinação do ILS com o RVND (Hansen et al., 2008). Os resultados obtidos pelos algoritmos propostos são comparados com os melhores resultados encontrados pelo algoritmo SA* de Ruiz-Torres et al. (2013).

O restante do artigo tem a seguinte estrutura: na seção 2 é definido o problema, ilustrado um exemplo e apresentado o modelo matemático. A seção 3 apresenta as descrições dos algoritmos

ILS e ILS-RVND. As instâncias e os resultados dos algoritmos são mostrados na seção 4 e as conclusões do trabalho são apresentadas na seção 5.

2. Definição do Problema

O presente problema consiste em processar tarefas em máquinas levando em conta que cada tarefa executada provoca um certo desgaste na máquina. Este desgaste irá diminuir o desempenho da máquina, aumentando o tempo de processamento da tarefa seguinte a ser efetuada. A posição de uma tarefa na sequência é muito significativa para o rendimento total da máquina, ou seja, o desgaste da máquina depende da ordem das tarefas.

O problema é definido por Ruiz-Torres et al. (2013) da seguinte forma: existem n tarefas, estabelecendo o conjunto $N = \{1, \ldots, j, \ldots, n\}$, para serem processadas em m máquinas paralelas, que formam o conjunto $M = \{1, \ldots, k, \ldots, m\}$. Todas as tarefas são não-preemptivas[1] e estão disponíveis para o processamento no tempo zero. Cada máquina pode processar somente uma tarefa de cada vez, não pode ficar ociosa até a última tarefa atribuída ter sido finalizada e estas máquinas são não-relacionadas[2]. Além disso, o tempo de processamento da tarefa j na máquina k é definido por p_{jk} e o efeito de desgaste da tarefa j na máquina k é dado por d_{jk}, em que, $0 \leq d_{jk} \leq 1 \; \forall j \in N$ e $k \in M$.

Neste problema busca-se sequenciar as tarefas nas máquins a fim de minimizar o máximo tempo de conclusão do processamento das tarefas, também conhecido como *makespan* (C_{max}). É importante destacar que este problema é NP-difícil para $m \geq 2$ (Ruiz-Torres et al., 2013).

Dada uma solução, ou seja, um sequenciamento das tarefas nas máquinas, o desempenho da máquina k na posição h (isto é, após executar a tarefa que está na posição $h-1$) é determinado por:

$$q_{k[h]} = (1 - d_{[h-1]k}) * q_{k[h-1]} \tag{1}$$

onde, $q_{k[1]} = 1, \forall k \in M$ (ou seja, inicialmente as máquinas possuem 100% de desempenho) e $d_{[h]k}$ é o desgaste provocado na máquina k pela execução da tarefa que está na posição h. E o tempo real de processamento de uma tarefa j na máquina k é determinado por: $\frac{p_{jk}}{q_{kh}}$.

De acordo com a classificação de Graham et al. (1979), este problema será denotado $Rm|Sdd|C_{max}$ em que Rm significa o ambiente das máquinas não-relacionadas, Sdd corresponde às restrições de desgaste dependentes da sequência e C_{max} é relativo ao critério de otimização.

2.1. Exemplo numérico

A seguir é apresentado um exemplo para uma instância do problema com $n = 8$ e $m = 3$ A Tabela (1) apresenta os valores de tempos de processamento das tarefas (p_{jk}) nas respectivas máquinas e as porcentagens dos desgastes (d_{jk}) de cada tarefa em cada máquina.

Tabela 1: Exemplo de instância para 8 tarefas e 3 máquinas

Tarefa	p_{j1}	p_{j2}	p_{j3}	d_{j1}	d_{j2}	d_{j3}
1	26,5	63,5	65,5	0,04	0,01	0,01
2	20,0	27,9	46,6	0,03	0,02	0,03
3	30,5	47,9	62.4	0,02	0,02	0,01
4	31,1	22,4	80.0	0,02	0,02	0,01
5	82,8	77,4	92.3	0,03	0,04	0,04
6	50,0	90,6	99.3	0,01	0,04	0,01
7	56,4	28,2	76.2	0,03	0,03	0,01
8	21,8	42,3	24.5	0,02	0,03	0,03

[1] Não-preemptivo significa que a execução de cada tarefa não pode ser interrompida até que seja totalmente concluída.

[2] Em máquinas não-relacionadas cada tarefa j possui um tempo de processamento que depende da máquina k na qual será alocada.

Na Figura 1 mostra-se um sequenciamento no qual as tarefas 2, 6 e 3 são atribuídas aleatoriamente para a máquina 1, as tarefas 7, 4 e 5 para a máquina 2 e as tarefas 8 e 1 para a máquina 3. Para encontrar o tempo de processamento real das tarefas é necessário calcular o desempenho da máquina após a execução de uma tarefa. Na Figura 1, observa-se que as máquinas M_1, M_2 e M_3, na primeira posição, executam as tarefas 2, 7 e 8, respectivamente, com 100% de desempenho. Estas primeiras tarefas causam certo desgaste diminuindo o desempenho da máquina. Por exemplo, a tarefa 7 provocou um desgaste na máquina 2, então o novo desempenho desta máquina na posição $h = 2$ (após executar a tarefa 7) é obtido pela equação (1) da seguinte maneira: $q_{2[2]} = (1 - d_{[1]2}) * q_{2[1]} = (1 - 0,03) * 1 = 0,97 = 97\%$. Como o desempenho da máquina diminuiu, então a tarefa que ocupa a segunda posição será executada em um tempo maior. O tempo real de processamento para a tarefa 4, que ocupa a posição $h = 2$ na máquina 2, é obtido da seguinte maneira: $\frac{22,4}{0,97} = 23,1$. Na Figura 1 este tempo está entre parênteses dentro do retângulo que representa a tarefa. Isto significa que, após uma máquina k executar uma tarefa j, o desempenho da máquina cai e o tempo de processamento da tarefa aumenta. Desta forma, determina-se os tempos reais de processamento de todas as tarefas e o desempenho das máquinas, após executar uma tarefa. Ainda na Figura 1, os valores mostrados na linha do tempo são os tempos finais do processamento das tarefas, ou seja, o instante de conclusão das tarefas. O *makespan* é o maior instante de conclusão que corresponde à máquina 2, o qual é igual a 132,8.

Ruiz-Torres et al. (2013) demonstrou que, para determinar o menor tempo de conclusão de um conjunto de tarefas J alocado a uma máquina k, as tarefas deste conjunto devem estar ordenadas, em ordem decrescente, pela seguinte relação:

$$r_{jk} = p_{jk}(1 - d_{jk})/d_{jk}, \forall j \in J, \forall k \in M \quad (2)$$

Esta relação determina que, geralmente as tarefas com os maiores tempos de processamento e com os menores desgastes devem ser processadas primeiras.

Na Figura 2 apresenta-se o sequenciamento das tarefas reordenadas pela relação r_{jk}. Observa-se que os tempos finais do processamento das tarefas em todas as máquinas são menores em comparação ao exemplo da Figura 1.

Conclui-se que no problema abordado basta determinar o conjunto de tarefas J a serem alocadas em cada máquina. A ordenação das tarefas será determinada pela relação r_{jk}.

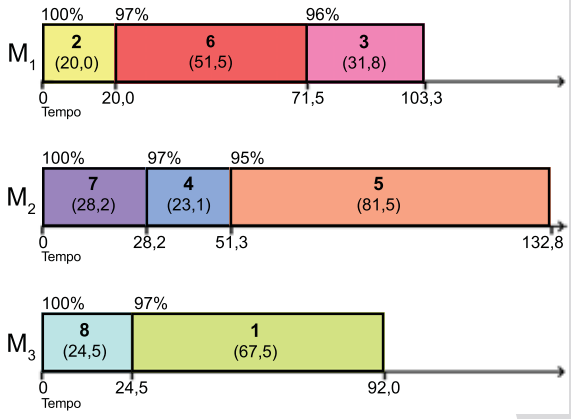

Figura 1: Tarefas ordenadas aleatoriamente

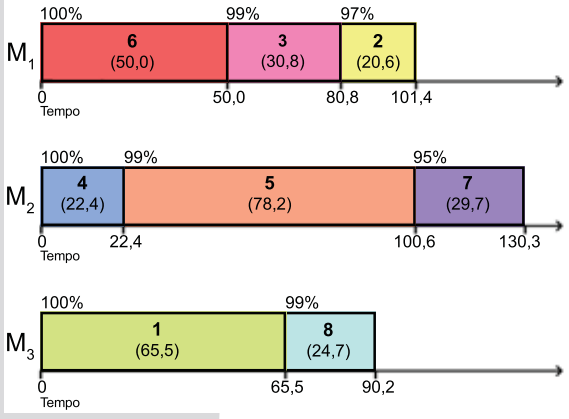

Figura 2: Tarefas ordenadas por r_{jk}

2.2. Formulação matemática

Esta seção apresenta um modelo de programação não-linear inteira do problema $Rm|Sdd|C_{max}$, proposto em Ruiz-Torres et al. (2013). Neste modelo usam-se as seguintes variáveis de decisão:

- C_{max} define o *makespan*.
- q_{kh} é o desempenho da máquina k após executar a tarefa da posição h.

- $x_{jkh} = \begin{cases} 1 & \text{, se a tarefa } j \text{ é atribuída para a máquina } k \text{ na posição } h, \\ 0 & \text{, caso contrário.} \end{cases}$

Denota-se, ainda, G como o conjunto das possíveis posições h.
O modelo matemático do problema $Rm|Sdd|C_{max}$ é dado por:

$$\min C_{max} \tag{3}$$

$$\sum_{j \in N} x_{jkh} \leq 1, \forall h \in G, k \in M \tag{4}$$

$$\sum_{h \in G} \sum_{k \in M} x_{jkh} = 1, \forall j \in N \tag{5}$$

$$\sum_{j \in N} \sum_{h \in G} \frac{p_{jk}}{q_{kh}} * x_{jkh} \leq C_{max}, \forall k \in M \tag{6}$$

$$\sum_{l \in N} x_{lk(h-1)} \geq x_{jkh}, \forall j \in N, k \in M, h \in G \backslash \{1\} \tag{7}$$

$$\sum_{j \in N} (1 - d_{jk}) * q_{k(h-1)} * x_{jk(h-1)} = q_{kh}, \forall h \in G \backslash \{1\}, k \in M \tag{8}$$

$$q_{k1} = 1, \forall k \in M \tag{9}$$

$$x_{jkh} \in \{0, 1\}, \forall j \in N, k \in M, h \in G \tag{10}$$

No modelo, o objetivo é minimizar o C_{max} (*makespan*) definido na equação (3). A restrição (4) garante que cada posição em cada máquina pode ter no máximo uma tarefa atribuída e a (5) garante que cada tarefa deve ser atribuída para exatamente uma posição em uma máquina. A restrição (6) estabelece que a soma total dos processamentos em cada máquina não deve ser maior que C_{max}. Enquanto a (7) garante atribuições de tarefas contínuas. As equações (8) e (9) definem o nível de desempenho em cada posição da máquina. Sendo que para a primeira posição, todas as máquinas têm desempenho igual a um, ou seja, 100% de produtividade. Por fim, a equação (10) define os valores binários para as variáveis x_{jkh}.

Pode-se perceber que este modelo de programação inteira é não-linear por causa da expressão $\frac{x_{jkh}}{q_{kh}}$ na restrição (6). Esta expressão não é tão simples de linearizar, assim, não foi possível resolver o modelo através do software CPLEX.[3]

3. Heurísticas Propostas

Neste trabalho, para resolver o problema $Rm|Sdd|C_{max}$ foram desenvolvidas duas heurísticas, uma baseada na meta-heurística *Iterated Local Search* (ILS) e a outra obtida pela combinação do ILS com o *Randon Variable Neighboard Descendent* (ILS-RVND). Esta seção apresenta uma descrição destas heurísticas.

3.1. ILS e ILS-RVND

Os passos da heurística ILS são ilustrados no Algoritmo 1. Este algoritmo recebe como entrada os parâmetros $CriterioDeParada$ e t. O primeiro parâmetro armazena o tempo máximo de execução utilizado como critério de parada do ILS. O parâmetro t é usado para definir o tamanho da perturbação. Os valores destes parâmetros serão estabelecidos na calibração de parâmetros na seção (4.2). No início do Algoritmo 1 determina-se uma solução inicial utilizando o método SOLUCAO_INICIAL e essa solução é melhorada pelo procedimento BUSCA_LOCAL, obtendo a melhor solução atual (S_{Best}). Em seguida é inicializado o laço de repetição, onde, primeiramente,

[3]CPLEX é um software utlizado para resolver modelos matemáticos com restrições lineares ou quadráticas, função objetivo linear e variáveis contínuas ou inteiras (Entringer e Arica, 2013).

a função de PERTURBACAO é chamada. A perturbação produz alterações na solução S_{Best} a fim de sair dos ótimos locais e gerar soluções novas próximas dos ótimos globais. A solução perturbada ($S_{Perturbada}$) é melhorada pelo procedimento da BUSCA_LOCAL. A solução aprimorada ($S_{melhorVizinho}$) retornada pela busca local é comparada com a melhor das soluções encontradas até o momento (S_{Best}). Desta forma, se a função objetivo do melhor vizinho for menor que a atual, a melhor solução é atualizada. Logo após, o critério de parada do laço é verificado, se este for satisfeito, o laço é finalizado, a melhor solução de todas (S_{Best}) é retornada e o algoritmo finaliza. Caso contrário, o laço é executado novamente.

Algoritmo 1: ILS($CriterioDeParada, t$)

```
1  início
2      S ← SOLUCAO_INICIAL( )
3      S_Best ← BUSCA_LOCAL(S)
4      repita
5          S_Perturbada ← PERTURBACAO(S_Best, t)
6          S_melhorVizinho ← BUSCA_LOCAL(S_Perturbada)
7          se f(S_melhorVizinho) < f(S_Best) então
8              S_Best ← S_melhorVizinho
9          fim
10     até CriterioDeParada é satisfeito;
11     retorna S_Best
12 fim
```

Além do ILS, é desenvolvido outro algoritmo similar que apresenta como diferença o procedimento BUSCA_LOCAL. Este novo algoritmo é denotado como ILS-RVND, pois utiliza a metaheurística *Randon Variable Neighboard Descendent* (RVND) como procedimento de BUSCA_LOCAL.

A seguir são apresentadas as descrições dos métodos utilizados nos algoritmos: SOLUCAO_INICIAL, BUSCA_LOCAL, RVND e PERTURBACAO.

3.2. Solução Inicial

A heurística utilizada para construir uma solução inicial é apresentada no Algoritmo 2. Inicialmente ordenam-se as tarefas de acordo com uma regra de prioridade, obtendo uma *Lista* de tarefas. Para cada tarefa da *Lista*, procura-se a melhor máquina para ser inserida, ou seja, a máquina que produza o menor tempo de conclusão. As tarefas são inseridas nas máquinas nas posições determinadas pela equação (2). O algoritmo finaliza quando todas as tarefa da *Lista* ordenada são inseridas nas máquinas, isto é, a *Lista* estiver vazia. São utilizadas 9 regras de prioridade para ordenar as tarefas, em ordem decrescente. As 8 primeiras foram propostas por Ruiz-Torres et al. (2013) e a última foi proposta neste trabalho. As regras utilizam as seguintes medidas para ordenar as tarefas:

- $p_j^{min} = min_{k \in M} \{p_{jk}\} \; \forall j \in N$ (usada na Regra 1).
- $p_j^{max} = max_{k \in M} \{p_{jk}\} \; \forall j \in N$ (usada na Regra 2)
- $d_j^{min} = min_{k \in M} \{d_{jk}\} \; \forall j \in N$ (usada na Regra 3).
- $d_j^{max} = max_{k \in M} \{d_{jk}\} \; \forall j \in N$ (usada na Regra 4).
- $r_j^{min} = min_{k \in M} \{r_{jk}\} \; \forall j \in N$ (usada na Regra 5).
- $r_j^{max} = max_{k \in M} \{r_{jk}\} \; \forall j \in N$ (usada na Regra 6).
- $v_j^{min} = min_{k \in M} \{p_{jk}/(1 - d_{jk})\} \; \forall j \in N$ (usada na Regra 7).
- $v_j^{max} = max_{k \in M} \{p_{jk}/(1 - d_{jk})\} \; \forall j \in N$ (usada na Regra 8).
- $r_j^{medio} = \frac{\sum_{k \in M} p_{jk}(1-d_{jk})/d_{jk}}{m} \; \forall j \in N$ (usada na Regra 9).

Para obter uma solução inicial do ILS e do ILS-RVND a heurística é executada 9 vezes, cada execução com uma regra diferente. Das 9 soluções geradas, a melhor é utilizada como solução inicial.

Algoritmo 2: SOLUCAO_INICIAL()

```
1  início
2      para cada regra faça
3          Lista ← OrdenaçãoRegraDePrioridade(regra)
4          enquanto Lista ≠ ∅ faça
5              tarefa ← primeira tarefa da Lista
6              Escolher melhor máquina para a tarefa
7              Inserir tarefa na melhor posição da máquina k
8              Lista ← Lista - {tarefa}
9          fim
10         Seja S a solução obtida
11     fim
12     retorna a melhor solução S
13 fim
```

3.3. Busca Local e procedimento RVND

A busca local tenta melhorar uma solução através de uma busca em vizinhança. As estruturas de vizinhanças utilizadas na busca local são definidas a seguir:

- N_1 (*Troca*): é a vizinhança formada por soluções obtidas realizando trocas de tarefas que estão na máquina com maior tempo de conclusão e tarefas das outras máquinas.
- N_2 (*Inserção*): esta vizinhança é formada por soluções obtidas retirando tarefas da máquina com maior tempo de conclusão e inserindo-as em outra máquina.

O número total de vizinhanças analisado pela vizinhança N_1 é dado por $n_s(n - n_s)$, e para N_2 é igual a $n_s(m-1)$, em que n_s representa o total de tarefas na máquina com o maior tempo de conclusão (Ruiz-Torres et al., 2013).

Os passos da busca local utilizada pelo ILS são apresentados no Algoritmo 3. O procedimento recebe uma solução S. A partir desta solução determinam-se as melhores soluções vizinhas nas vizinhanças N_1 e N_2. A partir dessas duas soluções vizinhas seleciona-se a melhor (S_{melhor}). Esta solução é comparada com a solução corrente S. Caso exista melhoria, atualiza-se S e repete-se o procedimento. Caso contrário, o melhor vizinho encontrado é retornado e o procedimento é finalizado.

Algoritmo 3: BUSCA_LOCAL(S)

```
1  início
2      enquanto melhorou = true faça
3          S_Vizinho1 ← N1(S) // retorna o melhor vizinho ∈ N1
4          S_Vizinho2 ← N2(S) // retorna o melhor vizinho ∈ N2
5          S_melhor ← melhor(S_Vizinho1, S_Vizinho2)
6          se f(S_melhor) < f(S) então
7              S ← S_melhor
8          fim
9          senão
10             melhorou ← false
11         fim
12     fim
13     retorna S
14 fim
```

A seguir descreve-se o procedimento de busca local RVND utilizado pelo algoritmo ILS-RVND. Neste procedimento são usadas as mesmas vizinhanças N_1 e N_2, apresentadas anteriormente. No entanto, a ordem para serem exploradas é determinada aleatoriamente. O procedimento

RVND é apresentado no Algoritmo 4, o qual recebe uma solução S como entrada. Inicialmente forma-se a lista *LV* com as duas estruturas de vizinhanças: N_1 (*troca*) e N_2 (*inserção*). Aleatoriamente escolhe-se uma vizinhança N_i da lista e determina-se a melhor solução desta vizinhança (S_{melhor}). Se a solução S_{melhor} for melhor do que a solução atual, a solução atual S recebe a melhor solução e a lista *LV* é reinicializada com todas as vizinhanças. Caso contrário, remove-se a vizinhança N_i da lista *LV*. O algoritmo finaliza quando a lista de vizinhanças ficar vazia, isto é, quando analisa-se todas as vizinhanças e não é encontrada uma solução melhor.

Algoritmo 4: RVND(S)

```
1  início
2      Lista de Vizinhancas: LV ← {N_1, N_2}
3      enquanto LV ≠ ∅ faça
4          i ← Randon(1, 2)
5          S_melhor ← N_i(S)
6          se f(S_melhor) < f(S) então
7              S ← S_melhor
8              LV ← {N_1, N_2}
9          fim
10         senão
11             LV ← LV − {N_i}
12         fim
13     fim
14     retorna S
15 fim
```

3.4. Perturbação

A perturbação é baseada em realocações de tarefas em máquinas diferentes. Das m máquinas, aleatoriamente escolhe-se t por cento das máquinas, sendo que a máquina com maior tempo de conclusão deve ser escolhida. Suponha a seguinte lista das máquinas escolhidas: $L = \{M_1, M_2, ..., M_H\}$. Escolhe-se aleatoriamente uma tarefa de cada máquina. A tarefa escolhida da máquina M_1 é inserida na máquina M_2, a tarefa da máquina M_2 é inserida na máquina M_3, assim sucessivamente, e a tarefa da máquina M_H é inserida na máquina M_1.

Este tipo de movimento é chamado de *Ejection Chain* e é utilizado em outros tipos de problemas (De Assis, 2013).

4. Testes Computacionais

Nesta seção são apresentados os resultados obtidos pelo ILS e ILS-RVND e estes são comparados com a heurística SA* proposta por Ruiz-Torres et al. (2013). Vale ressaltar que Ruiz-Torres et al. (2013) fornecem os melhores resultados obtidos pela heurística SA*.

Os testes foram feitos em um computador Intel Core i7, CPU (4GHz), com 32 GB de RAM, sistema operacioal Ubuntu 14.04, 64 bits. Os algoritmos ILS e ILS-RVND foram implementados em C++ e compilados com Netbeans IDE 8.0.2. Os algoritmos foram executados com uma simples *thread*.

O desempenho de cada algoritmo é medido através do *Relative Percentage Deviation* (RPD) definido pela sequinte equação:

$$RPD = \frac{f_{\text{media}} - f_{\text{Best}}}{f_{\text{Best}}} \times 100\% \tag{11}$$

na qual f_{media} é a média das funções objetivos de todas as execuções do algoritmo e f_{Best} é o melhor valor encontrado entre os 3 algoritmos, sendo eles: SA*, ILS e ILS-RVND. É importante ressaltar que quanto menor valor do RPD melhor será a qualidade da solução.

4.1. Instâncias do Problema

As instâncias do problema $Rm|Sdd|C_{max}$ foram produzidas por Ruiz-Torres et al. (2013) e estão disponíveis em: `http://ruiz-torres.uprrp.edu/dm/`. Esses autores geraram instâncias com o número de tarefas $n \in \{20, 35, 50\}$ e número de máquinas $m \in \{4, 7, 10\}$. O tempo de processamento das tarefas (p_{jk}) foram gerados aleatoriamente, com distribuição uniforme, sobre dois tipos de intervalos: [1; 100] e [100; 200]. Os efeitos de desgaste das máquinas produzidas pelas tarefas (d_{jk}) também foram gerados aleatoriamente, com distribuição uniforme, sobre dois tipos de intervalos: [1%; 5%] e [5%; 10%]. Combinando os parâmetros: n, m, os dois tipos de p_{jk} e os dois tipos de d_{jk}, têm-se 36 configurações possíveis. Para cada configuração, 25 instâncias foram geradas. Portanto, um total de 900 instâncias foram testadas.

Vale ressaltar, que para cada algoritmo, foram realizadas 10 execuções com cada instância. Para cada algoritmo proposto tem-se como resultado a média das 10 execuções.

4.2. Parâmetros das heurísticas

Esta seção define os valores dos parâmetros utilizados nos algoritmos ILS e ILS-RVND. São analisados os parâmetros da perturbação (*t*) e do tempo necessário para execução dos algoritmos (*CriterioDeParada*) .

O parâmetro *t* define a porcentagem do número de máquinas que participam da realocação de tarefas. Os valores utilizados para *t* são: $0,3m$, $0,5m$, $0,8m$ e m, sendo m o número total de máquinas. A Figura 3 mostra o gráfico de médias resultante do teste Tukey da Diferença Honestamente Significativa (HSD) com nível de confiança de 95% para as configurações testadas para o parâmetro *t*. Observa-se que as melhores médias são obtidas com $t = 0,5m$ (50% do total de máquinas). Utilizando menos do que 50% ou mais, as médias dos resultados tendem a piorar.

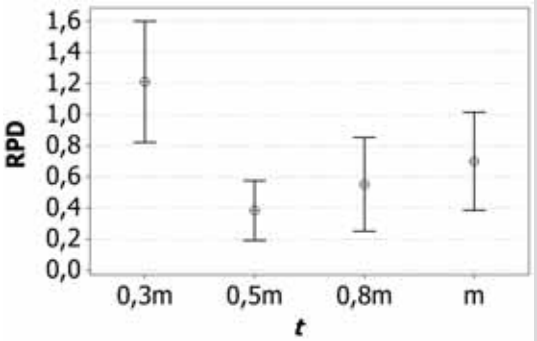

Figura 3: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração do parâmetro *t* do método de perturbação.

Além disso, é relevante analisar o tempo necessário de execução para os algoritmos ILS e ILS-RVND, que é definido como o critério de parada. A Figura 4 apresenta os gráficos de médias e intervalos HSD de Tukey com nível de confiança de 95%, para os resultados do ILS com os seguinte tempos de execução (em segundos): $\frac{0,25n}{m}$, $\frac{0,5n}{m}$, $\frac{n}{m}$, onde n representa o número de tarefas e m siginifica o total de máquinas das instâncias. Observa-se que quanto maior o tempo de execução, melhores são as soluções encontradas, pois os valores do RPD são menores. O mesmo pode ser percebido, na Figura 5, para os tempos de execução do algoritmo ILS-RVND. Para estes algoritmos, foi utilizado como condição de parada a execução de $\frac{n}{m}$ segundos.

4.3. Resultados

Os resultados obtidos pelo ILS e ILS-RVND são comparados com os melhores resultados do algoritmo SA* disponibilizados por Ruiz-Torres et al. (2013). É importante destacar que o SA* é composto de duas versões da meta-heurística *Simulated Annealing*, denominadas SA_1 e SA_2. SA_1 e SA_2 são executadas 8 vezes, cada execução inicia com uma solução obtida por um regra de prioridade (veja 3.2). Desta forma, o SA* considera a melhor solução das 16 execuções obtidas

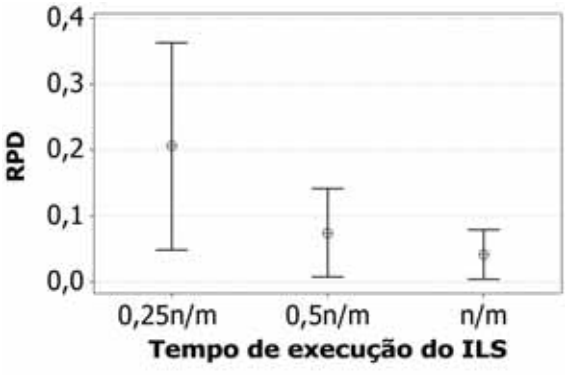

Figura 4: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para os tempos de execução do ILS.

Figura 5: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para os tempos de execução do ILS-RVND.

por SA_1 e SA_2. O critério de parada de SA_1 e SA_2 é estabelecido como $2n$ iterações sem melhora da melhor solução, em que n representa o número total de tarefas. Destaca-se, ainda, que para analisar os resultados dos algoritmos propostos neste trabalho, é utilizada a média de 10 execuções efetuadas e não o melhor valor encontrado. Ou seja, os resultados médios dos algoritmos ILS e ILS-RVND comparam-se com os melhores resultados fornecidos pelo algoritmo SA*.

Na Tabela 2 são exibidas as médias dos valores do RPD para os algoritmos comparados em cada grupo de instância. As 900 instâncias são agrupadas em 9 conjuntos de acordo com número de tarefas e de máquinas (*n x m*). Vale lembrar que o RPD é calculado considerando a melhor solução conhecida para cada instância, ou seja, os valores médios dos algoritmos ILS e ILS-RVND são comparados com os melhores valores conhecidos. Claramente pode-se notar que, para a maioria dos grupos de instâncias, os algoritmos propostos, ILS e ILS-RVND, apresentam as melhores médias em comparação ao SA*. Além disso, percebe-se que o algoritmo ILS-RVND apresenta um desempenho relativamente melhor que os outros dois (SA* e ILS) em todos os casos, com RPD médio de 0,16. Os melhores valores apresentam-se em negrito na Tabela 2. A última coluna da Tabela 2 exibe o tempo de execução gasto (em segundos) para os algoritmos propostos ILS e ILS-RVND. Para validar os resultados obtidos pelos algoritmos e verificar se as diferenças observadas

Tabela 2: Médias dos RPDs (por grupo de instâncias) dos algortimos comparados

Instância n x m	SA*	ILS	ILS-RVND	Tempo (s)
20 x 4	0,07	0,04	**0,01**	5
20 x 7	0,60	0,02	**0,01**	2
20 x 10	0,44	0,01	**0,00**	2
35 x 4	0,11	0,23	**0,01**	8
35 x 7	0,56	0,11	**0,04**	5
35 x 10	1,50	0,74	**0,32**	3
50 x 4	0,13	0,34	**0,03**	12
50 x 7	0,84	0,43	**0,25**	7
50 x 10	1,33	1,69	**0,81**	5
Média	0,62	0,40	**0,16**	5,44

são estatisticamente significantes, foi realizada uma Análise de Variância (ANOVA) paramétrica (Zar, 1984). Ainda, foram verificadas as três pressuposições da ANOVA para que os resultados do teste sejam estatisticamente válidos: normalidade, igualdade de variância e a independência dos resíduos. Dado que o *valor-P* na ANOVA resultou em 0,00 e este valor é menor que 0,05, pode-se concluir que as diferenças são estatisticamente significativas.

O gráfico de médias resultante do teste HSD de Tukey com nível de confiança de 95% para confrontar os algoritmos SA*, ILS e ILS-RVND é apresentado na Figura 6. Uma vez que o intervalo para o algoritmo SA* não sobrepõe os outros intervalos, a média do SA* é significativamente diferente das médias dos outros dois algoritmos. Isto é, as heurísticas propostas mostram desempenho superior ao método SA*.

Nota-se também na Figura 6 que o ILS-RVND tem desempenho significativamente melhor que o ILS, uma vez que não há sobreposição dos intervalos.

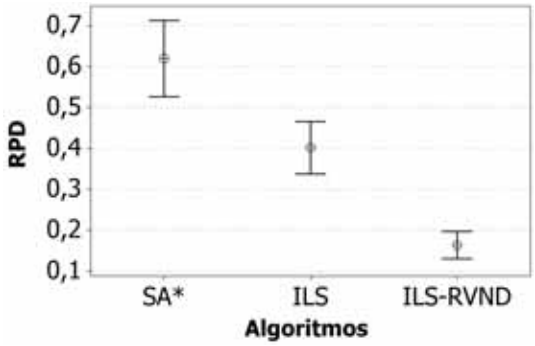

Figura 6: Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do algoritmo SA* e dos algoritmos propostos ILS e ILS-RVND.

É importante destacar, ainda, que executando 10 vezes cada instância, o tempo de CPU gasto, para execução das 900 instâncias, pelo algoritmo ILS (assim como pelo ILS-RVND) foi de aproximadamente: 13 horas. Os tempos de CPU gasto pelo SA* não são informados. Os autores do SA* somente informam que para um instância com $n = 14$ e $m = 4$, o SA* gasta 25 segundos e uma enumeração completa (força bruta) gasta 2 horas. Os algoritmos propostos, para resolver uma intância com $m = 50$ e $n = 4$, gastam 12,5 segundos.

5. Conclusões

Neste artigo estudou-se o problema de sequenciamento de tarefas em máquinas não-relacionadas, considerando que as tarefas provocam o desgaste das máquinas, o que diminui seus desempenhos e consequentemente eleva o tempo de conclusão das tarefas. Uma vez que o problema $Rm|Sdd|C_{max}$ é NP-difícil, neste artigo são propostos dois algoritmos heurísticos para encontrar soluções próximas da ótima em tempo computacional razoável. O primeiro algoritmo é baseado na meta-heurística ILS e o segundo é um método híbrido que combina o ILS com a busca local baseado no RVND.

Os resultados obtidos pelos algoritmos propostos são comparados com resultados gerados pelo algoritmo SA* proposto Ruiz-Torres et al. (2013). Os testes computacionais mostraram que o ILS e ILS-RVND apresentaram desempenhos superiores em comparação ao algoritmo SA*. Além disso, o ILS-RVND gerou melhores resultados comparados ao ILS, mostrando a eficiência da busca local RVND no algoritmo ILS. Os resultados obtidos foram validados através de testes estatísticos. Acredita-se que os resultados apresentados são os melhores resultados que existem até o momento na literatura do problema $Rm|Sdd|C_{max}$.

Como trabalhos futuros pretende-se desenvolver novos métodos fundamentados em outras meta-heurísticas.

Agradecimentos

Os autores agradecem o apoio da CAPES, CNPQ e FAPEMIG.

Referências

Browne, S. e Yechiali, U. (1990). Scheduling deteriorating jobs on a single processor. *Operations Research*, 38(3):495–498.

Cheng, T. E., Lai, P.-J., Wu, C.-C. e e Lee, W.-C. (2009). Single-machine scheduling with sum-of-logarithm-processing-times-based learning considerations. *Information Sciences*, 179(18):3127–3135.

De Assis, L. P. (2013). *Investigação de Metaheurísticas Aplicadas ao Problema de Roteamento de Veículos Multiobjetivo com Coleta Opcional*. PhD thesis, Universidade Federal de Minas Gerais.

Entringer, T. C. e Arica, G. G. M. d. (2013). Introdução ao cplex©: Otimização de modelos matemáticos. *Conflict*.

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

Gupta, S. K., Kunnathur, A. S. e e Dandapani, K. (1987). Optimal repayment policies for multiple loans. *Omega*, 15(4):323–330.

Hansen, P., Mladenović, N. e e Pérez, J. A. M. (2008). Variable neighbourhood search: methods and applications. *4OR*, 6(4):319–360.

Huang, X. e Wang, M.-Z. (2011). Parallel identical machines scheduling with deteriorating jobs and total absolute differences penalties. *Applied Mathematical Modelling*, 35(3):1349–1353.

Ji, M. e Cheng, T. E. (2008). Parallel-machine scheduling with simple linear deterioration to minimize total completion time. *European Journal of Operational Research*, 188(2):342–347.

Ji, M. e Cheng, T. E. (2009). Parallel-machine scheduling of simple linear deteriorating jobs. *Theoretical Computer Science*, 410(38):3761–3768.

Ji, M., Hsu, C.-J. e e Yang, D.-L. (2013). Single-machine scheduling with deteriorating jobs and aging effects under an optional maintenance activity consideration. *Journal of Combinatorial Optimization*, 26(3):437–447.

Joo, C. M. e Kim, B. S. (2015). Machine scheduling of time-dependent deteriorating jobs with determining the optimal number of rate modifying activities and the position of the activities. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 9(1):JAMDSM0007–JAMDSM0007.

Lee, W.-C., Wang, W.-J., Shiau, Y.-R. e e Wu, C.-C. (2010). A single-machine scheduling problem with two-agent and deteriorating jobs. *Applied Mathematical Modelling*, 34(10):3098–3107.

Lourenço, H. R., Martin, O. C. e e Stützle, T. (2003). *Iterated local search*. Springer.

Ma, W.-M., Sun, L., Liu, S. e e Wu, T. (2015). Parallel-machine scheduling with delivery times and deteriorating maintenance. *Asia-Pacific Journal of Operational Research*.

Mosheiov, G. (2012). A note: Multi-machine scheduling with general position-based deterioration to minimize total load. *International Journal of Production Economics*, 135(1):523–525.

Ruiz-Torres, A. J., Paletta, G. e e Pérez, E. (2013). Parallel machine scheduling to minimize the makespan with sequence dependent deteriorating effects. *Computers & Operations Research*, 40(8):2051–2061.

Toksarı, M. D. e Güner, E. (2009). Parallel machine earliness/tardiness scheduling problem under the effects of position based learning and linear/nonlinear deterioration. *Computers & Operations Research*, 36(8):2394–2417.

Yang, D.-L., Cheng, T., Yang, S.-J. e e Hsu, C.-J. (2012). Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities. *Computers & Operations Research*, 39(7):1458–1464.

Yang, S.-J. (2011). Parallel machines scheduling with simultaneous considerations of position-dependent deterioration effects and maintenance activities. *Journal of the Chinese Institute of Industrial Engineers*, 28(4):270–280.

Zar, J. H. (1984). Biostatistical analysis. 2nd. *Prentice Hall USA*.