Um método baseado em *Adaptive Large Neighborhood Search* para resolução de um problema de sequenciamento em máquinas paralelas

Luciano Perdigão Cota
Programa de Pós-graduação em Engenharia Elétrica
Universidade Federal de Minas Gerais (UFMG)
CEP: 31.270-901 – Belo Horizonte – MG – Brasil
lucianocota@ufmg.br

Frederico Gadelha Guimarães
Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais (UFMG)
CEP: 31.270-901 – Belo Horizonte – MG – Brasil
fredericoguimaraes@ufmg.br

Fernando Bernardes de Oliveira
Departamento de Computação e Sistemas
Universidade Federal de Ouro Preto (UFOP)
CEP: 35931-008 – João Monlevade – MG – Brasil
fernando@decea.ufop.br

Marcone Jamilson Freitas Souza
Departamento de Computação
Universidade Federal de Ouro Preto (UFOP)
CEP: 35400-000 – Ouro Preto – MG – Brasil
marcone@iceb.ufop.br

RESUMO

Este trabalho trata o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência com o objetivo de minimizar o *makespan*. Este problema aparece em indústrias de diferentes áreas, como têxteis e químicas, e sua resolução é desafiadora dada a sua complexidade. Para sua resolução, é proposto um algoritmo baseado no método *Adaptive Large Neighborhood Search* com aprendizado em autômatos para adaptar as probabilidades dos métodos de remoção e inserção. Um dos principais métodos de inserção é inspirado no algoritmo Húngaro, o qual é capaz de resolver subproblemas na otimalidade em tempo polinomial. Nos experimentos computacionais foi utilizado um subconjunto de problemas-teste da literatura, e os resultados encontrados foram comparados com os de dois outros algoritmos da literatura. Considerando o contexto experimental, os resultados sugerem que o algoritmo proposto encontrou os melhores resultados em 93% dos casos. Isso pode indicar a eficiência do método, de acordo com as condições estabelecidas.

PALAVRAS CHAVE. Problema de sequenciamento de máquinas, *makespan*, Aprendizado em autômatos, *Adaptive Large Neighborhood Search*
Área Principal: IND – PO na Indústria

ABSTRACT

This work deals with the unrelated parallel machine scheduling problem with sequence-dependent setup times, with the objective of minimizing the makespan. This problem appears in different industries as textile and chemicals, and its resolution is challenging due to its complexity. It was proposed for its resolution an algorithm based on the Adaptive Large Neighborhood Search method with learning automata to adapt the probabilities of using removal and insertion methods. One of the main insertion methods is inspired on the Hungarian algorithm, that is able to solve subproblems optimally. In the computational experiments were used a subset of instances of literature. The results were compared with two other algorithms of literature. For the experimental context the results suggest that the proposed algorithm found the best results in 93% of cases. This can indicate the efficiency of the proposed algorithm to the established conditions.

KEYWORDS. Machine scheduling problem. Makespan. Learning automata. Adaptive Large Neighborhood Search
Main Area: IND – PO in Industry

1. Introdução

Este trabalho trata o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência (UPMSP-ST, sigla do inglês *unrelated parallel machine scheduling problem colorred with sequence-dependent setup times*). O objetivo para este problema é a minimização do *makespan*.

O UPMSP-ST é um problema de grande relevância porque está presente em diversos tipos de indústrias. Sua complexidade de resolução se deve ao fato de ele pertencer à classe de problemas $\mathcal{NP}$-difíceis [Ravetti et al., 2007]. Ele é uma generalização do problema de sequenciamento em máquinas paralelas idênticas sem tempos de preparação (ou *identical parallel machine scheduling problem without setup times*), o qual foi demonstrado pertencer à classe de problemas $\mathcal{NP}$-difíceis em Karp [1972].

No UPMSP-ST tem-se um conjunto de tarefas $N = \{1, ..., n\}$, e um conjunto de máquinas $M = \{1, ..., m\}$, com as seguintes características: i) cada tarefa $j \in N$ deve ser alocada a uma única máquina $i \in M$; ii) o tempo para processar $j \in N$ em uma dada máquina $i \in M$ é dado por p_{ij} (tempo de processamento); iii) tem-se um tempo de preparação S_{ijk} para processar a tarefa $k \in N$ após a tarefa $j \in N$ na máquina $i \in M$, nesta ordem. O objetivo é alocar todas as n tarefas nas m máquinas com o propósito de minimizar o tempo máximo de conclusão do sequenciamento, definido como o *makespan*. O UPMSP-ST pode ser representado matematicamente como $R_M|S_{ijk}|C_{\max}$, segundo Graham et al. [1979]. Nessa representação, R_M indica máquinas independentes, S_{ijk} os tempos de preparação e $C_{\max}$ o *makespan*.

Na literatura são encontrados diversos trabalhos abordando o UPMSP-ST. Alguns desses trabalhos são apresentados a seguir. Em Al-Salem [2004] é desenvolvido um algoritmo chamado *partitioning heuristic* que combina três heurísticas: uma heurística construtiva, uma busca local, e uma heurística baseada no Problema do Caixeiro Viajante Assimétrico. Uma meta-heurística Meta-RaPS (*Metaheuristic for Randomized Priority Search*), um modelo matemático e um conjunto de problemas-teste são propostos em Rabadi et al. [2006]. Esses problemas-teste são disponibilizados em Scheduling Research [2005]. Em Arnaout et al. [2010] é proposto um algoritmo Colônia de Formigas. Vallada e Ruiz [2011] propõem dois algoritmos genéticos e um modelo matemático bem próximo ao de Rabadi et al. [2006]. Além disso, é desenvolvido um novo conjunto de problemas-teste disponibilizados em SOA [2011]. Um algoritmo Colônia de Abelhas é proposto por Lin e Ying [2014]. Rosales et al. [2013, 2015] apresentaram melhorias no modelo matemático de Vallada e Ruiz [2011]. A modificação no modelo permite que problemas-teste com até 60 tarefas sejam resolvidos de maneira ótima, considerando um tempo médio de execução entre 2 a 3 horas. Uma evolução do algoritmo de Colônia de Formigas proposto em Arnaout et al. [2010] foi desenvolvida por Arnaout et al. [2014]. O novo algoritmo é chamado ACOII. Nos trabalhos Haddad et al. [2014]; Cota et al. [2014a,b]; Haddad et al. [2015] foram propostos diferentes algoritmos baseados em combinações das meta-heurísticas *Iterated Local Search* (ILS) [Lourenço et al., 2003] e *Variable Neighborhood Descent* (VND) [Hansen et al., 2008]. O algoritmo AIRP, desenvolvido por Cota et al. [2014a], foi o que apresentou o melhor desempenho nos problemas-teste disponibilizados em SOA [2011], considerando as condições experimentais.

Os modelos matemáticos existentes ainda não conseguem resolver otimamente problemas de grande porte do UPMSP-ST em tempo de computação aceitável, como pode ser verificado em Rosales et al. [2015]. Dada essa dificuldade, o objetivo deste trabalho é propor um algoritmo adaptativo que resolva de maneira eficiente problemas de grande porte em um tempo restrito.

Neste trabalho é proposta a implementação da meta-heurística *Adaptive Large Neighborhood Search* (ALNS) [Ropke e Pisinger, 2006]. O ALNS possui um conjunto de heurísticas responsáveis por fazer inserções e remoções na solução corrente num processo de perturbação. O intuito desse processo é explorar o espaço de busca de maneira eficiente. É proposto neste trabalho a utilização de aprendizado em autômatos (LA, do inglês *Learning Automata*) para aplicar de maneira adaptativa os métodos de remoção e inserção. O funcionamento básico do ALNS proposto é

apresentado a seguir em cinco passos. No primeiro, i, uma solução inicial é gerada a partir de uma heurística construtiva inspirada na fase construtiva do *Greedy Randomized Adaptive Search Procedure* (GRASP) [Feo e Resende, 1995]. Em ii, inicia-se um laço de iterações. No passo iii, a cada iteração são aplicadas remoções e inserções de tarefas usando os métodos apropriados. Cada um desses métodos é selecionado por um procedimento do tipo roleta, que utiliza as probabilidades de cada método. Em iv, as buscas locais são realizadas pelo método *Random Variable Neighborhood Descent* (RVND) [Souza et al., 2010]. No passo v, ao final de cada iteração, o desempenho dos métodos de remoção e inserção são contabilizados por meio da LA.

Um dos principais métodos de inserção utilizados é inspirado no Algoritmo Húngaro. Este algoritmo é baseado nas ideias de König e Egerváry [Jungnickel, 2008], e foi introduzido por Kuhn [1955]. O Algoritmo Húngaro (AH) é capaz de resolver problemas de atribuição, os quais são definidos em Teoria dos Grafos como correlação ótima num grafo bipartido, em tempo polinomial. O método encontra uma atribuição ótima a partir de uma matriz de custos. Portanto, o método de inserção baseado no AH resolve subproblemas do UPMSP-ST de maneira ótima em tempo polinomial.

Nos experimentos computacionais o algoritmo proposto é executado em um conjunto de problemas-teste propostos por Vallada e Ruiz [2011] e disponibilizados em SOA [2011]. Os resultados obtidos são comparados aos dos algoritmos AIRP [Cota et al., 2014a] e GA2 [Vallada e Ruiz, 2011].

O restante deste trabalho é organizado como segue. Na Seção 2 é apresentado o aprendizado em autômatos. O algoritmo proposto é apresentado e detalhado na Seção 3. Na Seção 4 são descritos os experimentos computacionais. Na Seção 5 são apresentadas as conclusões e perspectivas de trabalhos futuros.

2. Aprendizado em autômatos

Um aprendizado em autômatos é uma unidade de decisão adaptativa. Ela melhora seu desempenho ao aprender a escolher a ação ideal a partir de um conjunto finito de ações permitidas por meio de interações repetidas num ambiente aleatório [Narendra e Thathachar, 1989]. Cada ação é escolhida aleatoriamente com base em uma distribuição de probabilidade mantida sob um conjunto de ações. Em cada instante, a ação escolhida é utilizada como entrada para o ambiente aleatório para aprendizagem futura.

Um aprendizado em autômatos pode ser definida por $(\phi, \alpha, \beta, A, G, P)$ [Narendra e Thathachar, 2012]. Cada um dos elementos é apresentado a seguir: i) ϕ: é um conjunto de estados internos; ii) α: é um conjunto de saídas ou ações; iii): β é um conjunto de respostas do ambiente; iv) A: é um aprendizado em autômatos; v) $G(.) : \phi \implies \alpha$: é uma função que mapeia o estado corrente (ϕ) para uma saída corrente (α); vi) e P: é um vetor de probabilidades que determina a probabilidade de seleção de um estado em cada estágio.

Ao sofrer uma ação, o ambiente aleatório responde a LA com um sinal de ruído (β). A LA utiliza essa resposta para ajustar as suas probabilidades de ações internas usando seu algoritmo de aprendizagem. Considere uma resposta do ambiente para uma ação α_i no passo k com $\beta \in \{0, 1\}$. Os números 0 e 1 são usados como respostas *agradáveis* e *desagradáveis*, respectivamente. Caso a resposta seja *agradável*, a probabilidade da ação interna pode ser atualizada de acordo com a Equação (1). Caso contrário, a Equação (2) é aplicada [Vafashoar e Meybodi, 2016].

$$p_j(k+1) = \begin{cases} p_j(k) + a(1 - p_j(k)) & \text{se } i = j \\ p_j(k)(1-a) & \text{se } i \neq j \end{cases} \quad (1)$$

$$p_j(k+1) = \begin{cases} p_j(k)(1-b) & \text{se } i = j \\ \frac{b}{r-1} + p_j(k)(1-b) & \text{se } i \neq j \end{cases} \quad (2)$$

Para as Equações (1) e (2), a e b são denominados parâmetros de *recompensa* e *penalidade*, respectivamente. O parâmetro r é o número de ações que podem ser escolhidas pelo autômato.

3. Algoritmo proposto

Esta seção apresenta o algoritmo ALNS proposto. Inicialmente, a representação e a avaliação da solução são descritas. Em seguida, o pseudocódigo do ALNS proposto é apresentado. Após isso, são descritos os métodos de remoção e inserção, bem como o procedimento de busca local.

3.1. Representação e avaliação da solução

Uma solução para problema é representada por um vetor de inteiros com m posições, sendo que m é o total de máquinas. A cada posição deste vetor está associada uma lista contendo as tarefas alocadas para a máquina correspondente. A avaliação de uma solução é realizado pelo seu valor de *makespan*.

3.2. Algoritmo ALNS

O algoritmo proposto é uma implementação baseada no método *Adaptive Large Neighborhood Search* [Ropke e Pisinger, 2006]. É incorporado ao algoritmo o aprendizado em autômatos como mecanismo adaptativo dos métodos de remoção e inserção. Duas estruturas de LAs são implementadas, sendo uma para os métodos de remoção (LA_{N^-}), e outra para os métodos de inserção (LA_{N^+}). Seis métodos de remoção e seis métodos de inserção são propostos. O procedimento para construir a solução inicial é inspirado na fase construtiva do GRASP. A busca local é realizada por meio do método RVND (*Random Variable Neighborhood Search*) [Souza et al., 2010]. O pseudocódigo do algoritmo é apresentado no Algoritmo 1.

O Algoritmo 1 possui os seguintes parâmetros de entrada: 1) $t_{\max}$: é o tempo máximo de execução; 2) $K = 6 \times \max(|\alpha^-|, |\alpha^+|)$: a cada K iterações, as probabilidades dos métodos de remoção e inserção são atualizadas pela LA; 3) $q = 0,05$: percentual de tarefas que são removidas e inseridas a cada iteração; 4) $a_1 = 0,2$: parâmetro de recompensa caso a solução encontrada seja melhor que a melhor solução; 5) $a_2 = 0,1$: parâmetro de recompensa caso a solução encontrada seja melhor que a solução corrente; 6) $a_3 = 0,05$: parâmetro de recompensa caso a solução encontrada seja aceita segundo o critério de temperatura; e 7) $b_1 = 0,02$: parâmetro de penalidade caso a solução encontrada não seja aceita. Os parâmetros K e q foram definidos previamente por meio de testes empíricos. A definição dos valores dos parâmetros a_1, a_2, a_3 e b_1 foram inspirados nas discussões realizadas em Vafashoar e Meybodi [2016].

A cada iteração do algoritmo são realizados os seguintes passos:

1. Um método de remoção $\alpha_i^- \in \alpha^-$ e um método de inserção $\alpha_j^+ \in \alpha^+$ é selecionado pelo procedimento da roleta utilizado as probabilidades de cada método;

2. O método de remoção α_i^- remove q tarefas da solução corrente (s'), depois estas tarefas são embaralhadas. Posteriormente, as q tarefas são inseridas na solução s' usando o método de inserção α_j^+. Os métodos de remoção e inserção funcionam como um procedimento de perturbação para explorar de maneira eficiente o espaço de busca;

3. É aplicado o procedimento de busca local $RVND$ na solução corrente s';

4. Em seguida, as probabilidades p^- e p^+ são atualizadas. Se a solução corrente s' é aceita, é aplicada a Eq. (1) usando os parâmetros de recompensa a_1, a_2 ou a_3. Caso a solução corrente s' não seja aceita, é aplicada a Eq. (2) usando o parâmetro de penalidade b.

5. A cada K iterações as probabilidades dos métodos de inserção e remoção são atualizadas. Esta atualização é feita periodicamente para que o aprendizado em autômatos tenha tempo de aprender com algumas ações no ambiente, e só depois, atualizar as probabilidades.

6. O algoritmo é encerrado quando é atingido o tempo máximo de execução ($t_{\max}$), e a melhor solução é retornada.

Algoritmo 1: ALNS

entrada: $t_{\max}, K, q, a_1, a_2, a_3, b_1$
saída : s_{melhor}

Defina LA $(\phi^-, \alpha^-, \beta, A^-, \pi^-, p^-)$;
Defina LA $(\phi^+, \alpha^+, \beta, A^+, \pi^+, p^+)$;
$s \leftarrow$ `procedimentoConstrutivo` ();
$s_{melhor} \leftarrow s$;
$T \leftarrow (0,1 \times f(s))/\ln 2$;
enquanto $tempoAtual \leq t_{\max}$ **faça**
  Selecione α_i^- usando o método da roleta;
  Selecione α_j^+ usando o método da roleta;
  Remova q tarefas de s usando α_i^-;
  Insira as tarefas removida de s usando α_j^+;
  $s' \leftarrow$ `RVND (`s`)` ;
  se $f(s') < f(s_{melhor})$ **então**
    $s_{melhor} \leftarrow s'$; $s \leftarrow s'$;
    $a = a_1$; $b = 0$;
    Atualize p^- e p^+ usando as Equações (1)-(2);
  fim
  senão se $f(s') < f(s)$ **então**
    $s \leftarrow s'$;
    $a = a_2$; $b = 0$;
    Atualize p^- e p^+ usando as Equações (1)-(2);
  fim
  senão se `aleatório` $< e^{-\frac{(f(s')-f(s))}{T}}$ **então**
    $s \leftarrow s'$;
    $a = a_3$; $b = 0$;
    Atualize p^- e p^+ usando as Equações (1)-(2);
  fim
  senão
    $a = 0$; $b = b_1$;
    Atualize p^- e p^+ usando as Equações (1)-(2);
  fim
  se $k \mod K = 0$ **então**
    `atualizaLA`(A^-, A^+, p^-, p^+);
  fim
  $T \leftarrow 0,99 \times T$;
fim
retorne s_{melhor}

3.3. Procedimento construtivo

Este procedimento é inspirado na fase construtiva do GRASP e na regra *Adaptive Shortest Processing Time* (ASPT) [Baker, 1974]. O seu funcionamento é apresentado a seguir. Inicialmente, é gerada uma solução inicial s utilizando o método construtivo guloso proposto na Seção II.D de Cota et al. [2014a]. A cada iteração é construída uma nova solução s' por meio de um método construtivo parcialmente guloso, descrito adiante. Se a solução s' for melhor que s, então s é substituída por s' ($s \leftarrow s'$). Caso contrário, s' é descartada. O processo é repetido até que o limite de tempo seja atingido, definido como 1% de $t_{\max}$. Ao final, a solução s é retornada.

O funcionamento do método parcialmente guloso é dado pelos passos a seguir: 1) Considere $N = \{1, ..., n\}$ o conjunto com todas as tarefas, e $M = \{1, ..., m\}$ o conjunto com todas as máquinas; 2) Todas as tarefas de N são inseridas na lista de candidatos LC; 3) Todos os pares (i, j) de máquina $i \in M$ e tarefa $j \in LC$ são classificados segundo uma função de avaliação g. Essa função calcula o custo de inserir cada tarefa j no final de cada máquina i, considerando os tempos de processamento e preparação; 4) Após isso, um par (i, j) é selecionado aleatoriamente entre os 20% melhores pares, ou seja, aqueles com menor valor da função g. A seguir, a tarefa j é inserida ao final da máquina i, e a tarefa j é removida de LC; 5) Os passos anteriores são repetidos até que todas as tarefas de LC sejam removidas.

3.4. Procedimentos de remoção e inserção

Nesta seção são apresentados os métodos de remoção e inserção utilizados como mecanismo de perturbação da solução corrente. Inicialmente, os métodos de remoção são apresentados.

1. Remoção aleatória: seleciona e remove q tarefas aleatoriamente da solução corrente;
2. Remoção maior custo: remove as q tarefas mais custosas. Para isso, utiliza-se como critério o somatório dos custos de processamento e preparação;
3. Remoção maior custo semigulosa: é uma versão semigulosa do método anterior. Selecionam-se aleatoriamente q tarefas entre 20% das tarefas de maior custo.
4. Remoção máquina aleatória: seleciona aleatoriamente uma máquina, e remove as q primeiras tarefas desta máquina. Caso a máquina selecionada possua um número de tarefas menor que q, outra máquina é selecionada da mesma maneira;
5. Remoção maior máquina: seleciona a máquina mais custosa, e remove as suas q primeiras tarefas. O custo da máquina é dado pelo somatório dos tempos de processamento e preparação das tarefas alocadas a ela. Caso a máquina selecionada possua um número de tarefas menor que q, outra máquina é selecionada da mesma maneira;
6. Remoção *Shaw*: este método utiliza critério de similaridade para remover as tarefas e foi proposto por Shaw [1998]. Uma tarefa j' é selecionada aleatoriamente; em seguida, é calculada a diferença desta tarefa para todas as outras. As q tarefas mais similares de j' são removidas. O critério de similaridade adotado é o somatório do custo de processamento e de preparação de uma tarefa na posição alocada.

A seguir, são apresentados os métodos de inserção. É importante destacar que as q tarefas são embaralhadas antes de serem inseridas.

1. Inserção gulosa: a cada iteração, uma das q tarefas é inserida na melhor posição possível de todas as máquinas, ou seja, a posição que proporciona menor custo à máquina;
2. Inserção semigulosa: versão semigulosa do método anterior. Para cada tarefa escolhe uma posição aleatória entre 20% das melhores posições;

3. Inserção *Lambda*: insere 50% das q tarefas de maneira aleatória e as outras 50% usando o método de inserção gulosa;

4. Inserção ILS: este método é baseado na perturbação proposta na seção III.J de Cota et al. [2014a]. Para cada tarefa de q, uma máquina é selecionada aleatoriamente, e a tarefa é inserida na melhor posição, isto é, a que proporciona um menor custo à máquina;

5. Inserção arrependimento: baseia-se na ideia de inserir primeiro as tarefas que possuem um maior custo de arrependimento. Este custo é calculado para cada uma das tarefas de q, e é dado pela diferença da melhor posição para a segunda melhor posição. A melhor posição é aquela que proporciona menor custo à máquina. A seguir, as tarefas com maior valor de arrependimento são inseridas primeiro;

6. Inserção Húngaro: este método utiliza o algoritmo Húngaro [Kuhn, 1955] para alocar as q tarefas removidas. Como o processo depende de uma matriz quadrada, o número de tarefas utilizadas em cada iteração é igual ao número de máquinas. O seu funcionamento é dado pelos cinco passos a seguir: i) para a solução corrente com m máquinas, seleciona-se a mesma quantidade m das primeiras tarefas removidas; ii) avalia-se a melhor posição de alocação de cada uma destas tarefas em todas as m máquinas; iii) armazena-se em uma matriz quadrada os melhores custos de alocação em cada máquina para cada uma das tarefas; iv) esta matriz é utilizada pelo algoritmo Húngaro para definir a alocação ótima; v) insere-se as tarefas usando as alocações definidas pelo algoritmo Húngaro. Este processo é repetido até que se tenha no mínimo m tarefas sem alocar. Se houver tarefas restantes, elas são inseridas pelo método de inserção gulosa.

3.5. Procedimento de busca local

A busca local é realizada por meio do método *Random Variable Neighborhood Descent* [Souza et al., 2010]. O RVND é um pouco diferente do clássico *Variable Neighborhood Descent* (VND) [Hansen et al., 2008]. Enquanto no VND as buscas locais são executadas em ordem sequencial, no RVND, elas são executadas em ordem aleatória cada vez que o método é invocado. Três estruturas de vizinhança são utilizadas, como descrito a seguir:

1. **Múltipla inserção –** $NS^{MI}(s)$: utiliza o movimento de realocar uma tarefa de uma máquina para outra posição da mesma máquina, ou realocar uma tarefa para outra posição de outra máquina;

2. **Trocas na mesma máquina –** $NS^{SSM}(s)$: efetua o movimento de troca de duas tarefas em uma mesma máquina;

3. **Trocas em máquinas diferentes –** $NS^{SDM}(s)$: efetua movimento de troca de duas tarefas de duas máquinas diferentes.

O método RVND explora o espaço de soluções utilizando três buscas locais, executadas em ordem aleatória a cada chamada. Cada uma das buscas locais utiliza uma das estruturas de vizinhança descritas anteriormente. Todas as buscas locais utilizam a estratégia *First Improvement*. As buscas locais são descritas a seguir:

1. **Busca Local** FI_{MI}: é a mesma proposta na seção III.G de Cota et al. [2014a], e utiliza a estrutura de vizinhança $NS^{MI}(s)$;

2. **Busca Local** FI_{SMD}: é a mesma proposta na seção 3.4.2 de Haddad et al. [2014], e utiliza a estrutura de vizinhança $NS^{SMD}(s)$;

3. **Busca Local** FI_{SSM}: utiliza a estrutura de vizinhança $NS^{SSM}(s)$, e seu funcionamento é apresentado a seguir. Inicialmente, as máquinas são classificadas em ordem decrescente de acordo com o tempo de conclusão de cada máquina. As máquinas são ordenadas da que tem o maior tempo de conclusão para a que tem o menor tempo de conclusão. Em seguida, são analisadas para cada máquina todas as trocas possíveis entre as suas tarefas. Um vizinho $s' \in NS^{SSM}(s)$ é aceito se o tempo de conclusão da máquina envolvida é reduzido. Se a solução for aceita, o processo é interrompido; caso contrário, a busca continua até que todas as máquinas tenham sido analisadas.

4. Experimentos computacionais

Esta seção apresenta os experimentos computacionais realizados para avaliar o desempenho do algoritmo ALNS proposto. Os resultados são comparados com os dos algoritmos AIRP [Cota et al., 2014a] e GA2 [Vallada e Ruiz, 2011].

Um subconjunto com 600 problemas-teste de grande porte disponibilizados em SOA [2011] foi utilizado para comparar os métodos. Estes problemas envolvem combinações de 50, 100 e 150 tarefas com 10, 15, 20, 25 e 30 máquinas.

O algoritmo ALNS foi implementado na linguagem JAVA utilizando a IDE Netbeans 8.0.2. Os algoritmos ALNS e AIRP foram executados em computador Core i7, 1.9 GHz, 6 GB de RAM e Sistema Operacional Windows 7. Para o algoritmo GA2 são considerados os resultados disponibilizados em Vallada e Ruiz [2011]. O GA2 foi executado em um computador Pentium 4 com 2 GB de RAM.

O critério de parada utilizado pelos três algoritmos é o limite de tempo de execução. Esse limite é definido por: $t_{\max} = n \times (m/2) \times 50$ milisegundos, sendo n o número total de tarefas e m o número total de máquinas, tal como apresentado em Vallada e Ruiz [2011].

Para efeito de comparação, considerando que o GA2 foi executado em um computador diferente dos demais, a conversão dos tempos foi adotada. Conforme apresentado em PassMark [2017], o computador utilizado nas execuções com o ALNS e o AIRP é 1,13 vezes mais rápido que aquele empregado no GA2. Por isso, o limite de tempo definido para o ALNS e o AIRP foi dividido pelo fator 1,13.

Para comparar os resultados foi utilizada como métrica o desvio percentual relativo (RPD, do inglês *Relative Percentage Deviation*). Esta métrica é definida pela Equação (3), e foi empregada também por Vallada e Ruiz [2011] e Cota et al. [2014a].

$$RPD_i = \frac{\bar{f}_i^{Alg} - f_i^*}{f_i^*} * 100 \tag{3}$$

O $\bar{f}_i^{Alg}$ corresponde à solução encontrada pelo algoritmo *Alg* para o problema-teste i. O f_i^* é a melhor solução de cada problema-teste disponibilizada em SOA [2011].

Os algoritmos ALNS e AIRP foram executados 5 vezes para cada problema-teste. Somente o resultado médio foi considerado, devido à característica estocástica dos algoritmos. Já para o algoritmo GA2 foram considerados os valores do RPD disponibilizados em Vallada e Ruiz [2011].

Na Tabela 1 são apresentados os resultados médios da métrica RPD para os três algoritmos. Cada uma das 15 combinações de máquinas e tarefas possui 40 problemas-teste.

Os melhores resultados estão destacados em negrito. Os valores negativos indicam que os resultados encontrados foram melhores que a melhor solução disponibilizada em SOA [2011]. Observa-se que em apenas um dos casos ($m = 15$ e $n = 150$) o ALNS não encontrou resultado melhor que os demais algoritmos. Nesse caso ainda, o ALNS empate com AIRP ($-3,64$). O ALNS encontrou melhores resultados em 93% dos problemas-teste, considerando os critérios experimentais adotados. Um *box plot* dos resultados é apresentado na Figura 1. Essa imagem sugere que o ALNS possui, de maneira geral, melhores resultados.

Tabela 1: Resultados para os algoritmos GA2, AIRP e ALNS.

Máquinas	Tarefas	GA2	AIRP	ALNS
	50	6,49	-1,07	**-2,11**
10	100	5,54	0,42	**-2,14**
	150	5,28	-1,15	**-4,05**
	50	9,2	-4,40	**-6,05**
15	100	7,32	-3,16	**-6,61**
	150	6,8	-3,64	-3,64
	50	9,57	-6,80	**-8,30**
20	100	8,59	-6,62	**-10,77**
	150	7,4	-6,85	**-8,54**
	50	8,1	-8,80	**-11,65**
25	100	8,07	-10,50	**-15,30**
	150	7,05	-9,72	**-13,31**
	50	9,4	-8,59	**-11,67**
30	100	7,9	-11,40	**-17,80**
	150	7,17	-11,73	**-16,16**

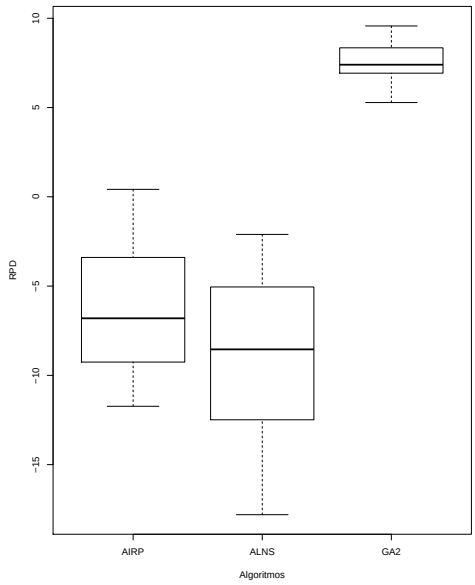

Figura 1: Resultados do box plot

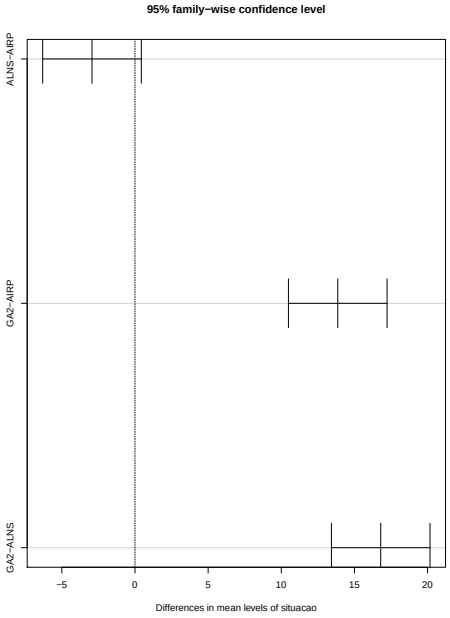

Figura 2: Resultados do teste Tukey HSD.

Testes foram empregados para identificar se existe diferença significativa entre os algoritmos. Inicialmente, foi utilizada a Análise de Variância (ANOVA) [Montgomery, 2007], considerando um grau de confiança igual a 95%. O resultado obtido por este teste para o $p - value$ é igual a $2,26 \times 10^{-15}$. Este valor sugere que existe uma diferença estatisticamente significativa entre os algoritmos.

Para identificar quais algoritmos apresentam essas diferenças, foi aplicado o teste Tukey HSD [Montgomery, 2007], considerando um grau de confiança igual a 95%. Os resultados deste teste são apresentados na Figura 2. Os resultados deste teste sugerem que os algoritmos ALNS e AIRP são estatisticamente diferentes do GA2, considerando o contexto experimental. Apesar da aparente superioridade do ALNS, seus resultados não são estatisticamente diferentes aos do AIRP. Entretanto, é possível observar que os resultados do algoritmo ALNS, considerando os critérios experimentais utilizados, são menores que os demais algoritmos na maioria dos casos.

5. Conclusões

Neste trabalho é tratado o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência, tendo como objetivo minimizar o *makespan*. É importante a sua resolução, pois este problema está presente em indústrias de diferentes áreas, e pertence à classe de problemas $\mathcal{NP}$-difíceis.

Para a resolução do problema, é proposta a implementação da meta-heurística *Adaptive Large Neighborhood Search* com aprendizado em autômatos para adaptar as probabilidades de aplicação dos métodos de remoção e inserção. A solução inicial é gerada por um procedimento construtivo, inspirado na fase construtiva do GRASP. O espaço de busca é explorado por meio de seis métodos de remoção e seis métodos de inserção. Um dos principais métodos de inserção é inspirado no algoritmo Húngaro. É importante notar que este método é capaz de resolver subproblemas de maneira ótima.

Nos experimentos computacionais, o algoritmo ALNS é executado para um subconjunto de problemas-teste. O seu desempenho foi comparados com dois algoritmos da literatura. O ALNS encontrou os melhores resultados em 93% dos casos. Considerando os critérios experimentais utilizados, os resultados sugerem uma diferença estatisticamente significativa com o GA2, e um desempenho aparentemente melhor que o AIRP. Entretanto, para este último, os testes estatísticos não sugerem diferença significativa.

Como trabalhos futuros, propõem-se a realização de um estudo sobre o desempenho de aprendizado em autômatos na fase adaptativa do ALNS, considerando cada um dos métodos de inserção e remoção, e a criação de uma versão multiobjetivo do algoritmo proposto.

Agradecimentos

Esta pesquisa foi apoiada pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq, pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES e pela Fundação de Apoio à Pesquisa do Estado de Minas Gerais – FAPEMIG.

Referências

Al-Salem, A. (2004). Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, 17(1):177–187.

Arnaout, J. P., Musa, R., e Rabadi, G. (2014). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines - part ii: enhancements and experimentations. *Journal of Intelligent Manufacturing*, 25:43–53.

Arnaout, J., Rabadi, G., e Musa, R. (2010). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 21(6):693–701.

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling*. John Wiley & Sons.

Cota, L. P., Haddad, M. N., Souza, M. J. . F., e Coelho, V. N. (2014a). AIRP: A heuristic algorithm for solving the unrelated parallel machine sheduling problem. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC 2014)*, p. 1855–1862, Beijing.

Cota, L. P., Haddad, M. N., Souza, M. J. F., e Martins, A. X. (2014b). A heuristic algorithm for solving the unrelated parallel machine scheduling problem with sequence dependent setup time (in portugueses). In *Proceedings of the 35th Brazilian Congress of Applied and Computational Mathematics*, Natal, Brazil. Available at http://proceedings.sbmac.org.br/sbmac/article/view/724/730.

Feo, T. e Resende, M. (1995). Greedy randomized search procedures. *Journal of Global Optimization*, 6:109–133.

Graham, R., Lawler, E., Lenstra, J., e Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete Mathematics*, 5(2):287–326.

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, Lisbon, Portugal. DOI: 10.5220/0004884603760383.

Haddad, M. N., Cota, L. P., Souza, M. J. F., e Maculan, N. (2015). Solving the unrelated parallel machine scheduling problem with setup times by efficient algorithms based on iterated local search. *Lecture Notes in Enterprise Information Systems*, 227:131–148.

Hansen, P., Mladenovic, N., e Pérez, J. A. M. (2008). Variable neighborhood search: methods and applications. *4OR: Quartely journal of the Belgian, French and Italian operations research societies*, 6:319–360.

Jungnickel, D. (2008). *Graph Networks and Algorithms*, volume 5 of *Algorithms and Computation in Mathematics*. Springer-Verlag Berlin Heidelberg, New York, 3 edition.

Karp, R. M. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40(4):85–103.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97.

Lin, S. W. e Ying, K. C. (2014). ABC-based manufacturing scheduling for unrelated parallel machines with machine-dependent and job sequence-dependent setup times. *Computers & Operations Research*, 51:172–181.

Lourenço, H. R., Martin, O., e Stützle, T. (2003). Iterated local search. In Glover, F. e Kochenberger, G., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, p. 321–353. Kluwer Academic Publishers, Norwell, MA.

Montgomery, D. (2007). *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, fifth edition.

Narendra, K. S. e Thathachar, K. S. (1989). *Learning Automata: An Introduction*. Prentice-Hall, New York.

Narendra, K. S. e Thathachar, M. A. (2012). *Learning automata: an introduction*. Courier Corporation.

PassMark (2017). Cpu benchmarks. URL `https://www.cpubenchmark.net/`. Acessado em 1 Fev 2017.

Rabadi, G., Moraga, R. J., e Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17(1):85–97.

Ravetti, M. G., Mateus, G. R., Rocha, P. L., e Pardalos, P. M. (2007). A scheduling problem with unrelated parallel machines and sequence dependent setups. *International Journal of Operational Research*, 2:380–399.

Ropke, S. e Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40:455–472.

Rosales, O. A., Alvarez, A., e Bello, F. A. (2013). A reformulation for the problem scheduling unrelated parallel machines with sequence and machine dependent setup times. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, p. 278–282, Rome, Italy.

Rosales, O. A., Bello, F. A., e Alvarez, A. (2015). Efficient metaheuristic algorithm and reformulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 76: 1705–1718.

Scheduling Research (2005). Scheduling research virtual center. A web site that includes benchmark problem data sets and solutions for scheduling problems. Disponível em `http://www.schedulingresearch.com`.

Shaw, P. (1998). *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems*, p. 417–431. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-49481-2.

SOA (2011). Sistemas de Optimizacion Aplicada. A web site that includes benchmark problem data sets and solutions for scheduling problems. Disponível em `http://soa.iti.es/problem-instances`.

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

Vafashoar, R. e Meybodi, M. R. (2016). Multi swarm bare bones particle swarm optimization with distribution adaption. *Applied Soft Computing*, 47:534 – 552. ISSN 1568-4946. URL `http://www.sciencedirect.com/science/article/pii/S1568494616303088`.

Vallada, E. e Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211 (3):612–622.