Proceeding Series of the Brazilian Society of Applied and Computational Mathematics, Vol. 3, N. 1, 2015.
Trabalho apresentado no XXXV CNMAC, Natal-RN, 2014.

Um algoritmo heurístico para resolver o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência

Luciano Perdigão Cota, **Matheus Nohra Haddad**,
Marcone Jamilson Freitas Souza, **Alexandre Xavier Martins**
Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Ouro Preto
35400-000, Ouro Preto, MG
E-mail: {lucianoufop, mathaddad}@gmail.com, marcone@iceb.ufop.br, xmartins@icea.ufop.br

Resumo: *Este trabalho trata o Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempos de Preparação Dependentes da Sequência, objetivando minimizar o makespan. É proposto um algoritmo heurístico na qual a solução inicial é gerada por meio de um método construtivo guloso, e depois refinada pelo procedimento Iterated Local Search (ILS). O ILS usa como busca local o procedimento Adaptive Local Search (ALS), desenvolvido neste trabalho. O ALS consiste de duas buscas locais e uma perturbação, sendo que cada uma delas é escolhida de acordo com uma certa probabilidade, a qual depende do sucesso em iterações pregressas. Os experimentos computacionais mostraram que o algoritmo proposto supera dois algoritmos da literatura, tanto em termos de qualidade quanto em variabilidade da solução final. Além disso, o algoritmo estabeleceu novos limites superiores em mais de 95% das instâncias.*

Palavras-chave: *Máquinas Paralelas, Makespan, Iterated Local Search, Adaptive Local Search*

1 Introdução

Este trabalho trata o problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempos de Preparação Dependentes da Sequência – UPMSPST, do inglês *Unrelated Parallel Machine Scheduling Problem with Setup Times*, tendo como objetivo minimizar o tempo total de conclusão do sequenciamento, o chamado *makespan*.

O UPMSPST tem importância tanto teórica quanto prática. A importância prática segue do fato de o UPMSPST estar presente em indústrias de diferentes ramos, como: téxteis, químicas, tintas, semicondutores e papéis [9]. Já a importância teórica se deve ao fato de o UPMSPST pertencer à classe $\mathcal{NP}$-difícil, visto ser ele uma generalização do *Parallel Machine Scheduling Problem with Identical Machines and without Setup Times* [4].

Encontrar uma solução ótima para problemas de grande porte utilizando métodos exatos pode ser computacionalmente inviável. Por isso, geralmente são utilizados procedimentos heurísticos para gerar soluções próximas à solução ótima [9].

No UPMSPST há um conjunto de máquinas não-relacionadas M e um conjunto de tarefas N, com as seguintes características: *i*) cada tarefa deve ser processada exatamente uma vez por uma máquina; *ii*) cada tarefa $j \in N$ tem um tempo de processamento p_{ij} para processar a tarefa $j \in N$ na máquina $i \in M$; *iii*) Dadas duas tarefas $j, k \in N$ existe um tempo de preparação S_{ijk} para processá-las, nesta ordem, na máquina $i \in M$; *iv*) Existe um tempo de preparação S_{i0j} para processar a primeira tarefa de uma máquina $i \in M$. O objetivo é minimizar o *makespan*.

Em [1] os autores propõem uma heurística de particionamento de fase, chamada *Partitioning Heuristic*, para resolver o UPMSPST. Em [9] é implementada uma metaheurística para busca randômica priorizada e uma formulação matemática para o problema. Já em [2], é desenvolvido

um método baseado em colônia de formigas para resolver um caso especial do problema, em que a proporção do número de tarefas e o número de máquinas é grande. Em [13] propõe-se um algoritmo *Simulated Annealing*, que reduz o esforço computacional eliminando movimentos de tarefas que não são promissoras. Em [12] são propostos dois algoritmos genéticos, chamados GA1 e GA2, que se diferenciam pelos valores dos parâmetros adotados. Esses autores também apresentaram uma formulação matemática para o problema, assim como criaram e disponibilizaram problemas-teste em [10]. Em [6] é desenvolvido um algoritmo heurístico denominado AIV, baseado no ILS e no *Random Variable Neighborhood Descent* – RVND [11]. Os autores mostraram que o algoritmo AIV é superior aos algoritmos de [12].

Neste trabalho é desenvolvido um algoritmo heurístico, denominado AIA, baseado na metaheurística *Iterated Local Search* – ILS [7], tendo como método de refinamento um procedimento de busca local adaptativa. Ele funciona da seguinte maneira: *i*) Constrói-se uma solução de forma gulosa utilizando a regra *Adaptive Shortest Processing Time* – ASPT; *ii*) Refina-se esta solução pelo ILS, utilizando como busca local uma heurística denominada ALS. O ALS consiste de duas buscas locais e uma perturbação, sendo que cada uma delas é escolhida de acordo com uma certa probabilidade, a qual depende do sucesso em iterações pregressas.

O AIA foi testado usando instâncias da literatura e os resultados computacionais mostraram que ele é capaz de produzir soluções melhores do que as dos algoritmos encontrados na literatura, e com menor variabilidade das soluções finais. Além disso, o algoritmo estabeleceu novos limites superiores para a maioria dos casos.

O restante deste trabalho está organizado como segue. A Seção 2 contém a descrição do algoritmo desenvolvido. Os experimentos computacionais estão relatados na Seção 3. Na Seção 4 conclui-se o trabalho apontando perspectivas de melhoramento do algoritmo desenvolvido.

2 Metodologia

2.1 Representação e Avaliação da Solução

Uma solução s é representada por um vetor v de m posições, em que cada posição representa uma máquina. A cada máquina, por sua vez, está associada uma lista composta das tarefas a ela alocadas. A solução s é avaliada pelo valor de *makespan*, isto é, pelo tempo de processamento da máquina mais sobrecarregada.

2.2 O Algoritmo AIA

O algoritmo AIA proposto neste trabalho, e apresentado pelo Algoritmo 1, possui dois parâmetros de entrada: *i*) $vezesNivel$, que representa o número de vezes de execução para cada nível de perturbação; *ii*) $tempoExec$, tempo em milisegundos para execução do algoritmo.

Na linha 1, a contabilização do tempo de execução é iniciada. Depois, na linha 2, são criadas 3 soluções para o problema, como segue: *i*) s, a solução corrente; *ii*) s', a solução modificada; *iii*) $melhorSol$, que armazena a melhor solução. Na linha 3 a solução s recebe a solução inicial criada pela regra *Adaptive Shortest Processing Time* – ASPT (ver subseção 2.3), a solução s passa por uma busca local utilizando o procedimento heurístico ALS (ver descrição na subseção2.5) e a $melhorSol$ recebe a solução s que passou pela busca local.

O nível de perturbação é inicializado com valor 1 na linha 4, e o tempo de execução é atualizado na linha 5. O processo iterativo do ILS inicia-se na linha 6 e termina na linha 26. Este processo iterativo é interrompido quando o limite de tempo é excedido.

Nas linhas 8 e 9 são inicializadas a variável que controla o número de vezes em que cada nível de perturbação ($nivel$) é aplicado, assim como o nível máximo de perturbações ($perturbMax$). Entre as linhas 12 e 15 são feitas as perturbações na solução corrente. Na linha 16, após a perturbação, a solução corrente passa pela busca local ALS. Entre as linhas 17 e 20 é analisado se as alterações feitas na solução corrente s' foram boas o suficiente para continuar a busca

DOI: 10.5540/03.2015.003.01.0412

a partir dela. Ao fim do tempo de execução, a variável *melhorSol* guarda a melhor solução encontrada. Quando o nível de perturbação é maior que 3 (Linha 25), ele é reiniciado com 1.

Nas demais subseções são detalhados cada módulo do algoritmo proposto.

Algoritmo 1: AIA

```
   entrada : vezesNivel, tempoExec
   saida   : melhorSol
 1 tempoAtual ← 0;
 2 Solução s, s', melhorSol;
 3 s ← ASPT(); s ← ALS(s); melhorSol ← s;
 4 nivel ← 1;
 5 Atualiza tempoAtual;
 6 enquanto tempoAtual ≤ tempoExec faça
 7 |   s' ← s;
 8 |   vezes ← 0;
 9 |   perturbMax ← nivel + 1;
10 |   enquanto vezes < vezesNivel faça
11 |   |   perturb ← 0; s' ← s;
12 |   |   enquanto perturb < perturbMax faça
13 |   |   |   perturb ++;
14 |   |   |   s' ← Perturb_ILS(s');
15 |   |   fim
16 |   |   s' ← ALS(s');
17 |   |   se f(s') < f(s) então
18 |   |   |   s ← s'; vezes ← 0;
19 |   |   |   atualizaMelhor(s, melhorSol);
20 |   |   fim
21 |   |   vezes ++;
22 |   |   Atualiza tempoAtual;
23 |   fim
24 |   nivel ++;
25 |   se nivel > 3 então nivel ← 1;
26 fim
27 retorne melhorSol ;
```

2.3 Adaptive Shortest Processing Time

O procedimento *Adaptive Shortest Processing Time* – ASPT é uma extensão da regra *Shortest Processing Time* [3] e tem por objetivo gerar uma solução inicial para o problema. Ele funciona como segue: Inicialmente, armazena-se em um conjunto N todas as tarefas e em um conjunto M todas as máquinas. Todas as tarefas de N são classificadas de acordo com uma função g, dada pela soma do tempo de processamento, do tempo de preparação e do tempo de conclusão, quando existirem. A ideia é inserir a tarefa na última posição da máquina que produz o menor tempo de conclusão. Se a máquina que tiver o menor tempo de conclusão ainda não tem alguma tarefa alocada, o novo tempo de conclusão da mesma será o tempo de preparação inicial da tarefa a ser inserida somando ao tempo de processamento de tal tarefa. Se esta máquina já possuir alguma tarefa, o novo tempo de conclusão da máquina será o tempo de conclusão anterior somado ao tempo de processamento da tarefa a ser inserida e ao tempo de preparação da última tarefa presente na máquina. O processo termina quando todas as tarefas forem alocadas a alguma máquina, gerando assim um solução factível. O procedimento é dito adaptativo porque a inserção de uma tarefa em alguma máquina depende das inserções anteriores.

2.4 Estruturas de Vizinhança

Para explorar o espaço de soluções são aplicados três tipos diferentes de movimentos, cada qual dando origem a um tipo de vizinhança, a saber: 1) Múltipla Inserção: Este movimento dá origem à vizinhança $N^{MI}(.)$, e consiste em realocar uma tarefa de uma máquina para qualquer outra posição na mesma máquina ou realocar esta tarefa para qualquer posição de outra máquina; 2) Trocas na Mesma Máquina: Este movimento, que dá origem à vizinhança $N^{T}MM(.)$, e consiste

em trocar de posição duas tarefas de uma mesma máquina; 3) Trocas em Máquinas Diferentes: Este movimento, que dá origem à vizinhança $N^TMM(.)$, consiste em trocar de posição duas tarefas de uma mesma máquina.

2.5 Procedimento ALS

O procedimento *Adaptive Local Search* – ALS é um procedimento que utiliza os métodos FI^1_{MI} e BI_{TMM} como buscas locais, e um método de perturbação chamado $Perturb\_TMD$. A ideia deste procedimento é aumentar, de forma adaptativa, a probabilidade dos métodos que tiverem o melhor desempenho em buscas preguessas.

Inicialmente todos os métodos (FI^1_{MI}, BI_{TMM} e $Perturb\_TMD$) têm a mesma probabilidade de ser escolhidos. A seguir, é executado um laço de repetições por 21 iterações sem melhoria na solução corrente. Em cada iteração um método é selecionado por um mecanismo de roleta utilizando a sua probabilidade. Quando o método selecionado proporciona uma melhoria na solução corrente o número de iterações é reinicializado.

A cada 7 iterações do laço de 21 iterações, as probabilidades de cada método são atualizadas. Para atualizar as probabilidades é calculada a média M_i do valor das soluções encontradas até o momento para cada método $i \in \{1, 2, 3\}$. A seguir, é verificada a "distância", dada por q_i, de cada média para a melhor solução $M^\star$ encontrada pelo algoritmo AIA, isto é, $q_i = M_i/M^\star$. De acordo com esta "distância" é calculada a nova probabilidade p_i de escolher o método i, calculada pela expressão $p_i = q_i/\sum_{j=1}^{3} q_j$. Quanto mais distante a média de um método estiver da melhor solução menor será sua probabilidade. Por outro lado, quanto mais próximo a "distância" da média de um método estiver da melhor solução maior será a sua probabilidade. Desta forma, o método que obtiver um melhor desempenho terá maior probabilidade de ser escolhido. Ao final é retornada a melhor solução encontrada.

2.5.1 Busca Local FI^1_{MI}

Esta busca local utiliza a vizinhança $N^MI(.)$ através da estratégia *First Improvement*. Ela funciona como segue: Inicialmente, as máquinas são ordenadas, em ordem decrescente, de acordo com a função custo de cada máquina. A seguir são aplicados movimentos envolvendo a máquina mais sobrecarregada com a máquina menos sobrecarregada. Um vizinho $s' \in N^MI(s)$ é aceito se: i) houver redução no custo das duas máquinas envolvidas (ou redução do custo na máquina envolvida, caso o movimento envolva uma única máquina); ii) houver melhora no custo em uma das máquinas e piora no custo na outra máquina, mas no cômputo geral, a melhoria for maior que a piora e não haja piora do *makespan*. Este critério é aplicado apenas para movimentos envolvendo duas máquinas. Em caso de aceitação, s' passa a ser a nova solução corrente s e este procedimento é reaplicado a partir da solução corrente. Caso contrário, outro vizinho s' é gerado. Quando forem analisados todos os movimentos de inserção envolvendo a máquina mais sobrecarregada e a menos sobrecarregada, passa-se a analisar os movimentos envolvendo a máquina mais sobrecarregada e a segunda menos sobrecarregada. Havendo uma melhora, reordenam-se as máquinas com relação à função custo e reinicia-se a busca. Caso contrário, passa-se a analisar movimentos envolvendo a máquina mais sobrecarregada e a terceira menos sobrecarregada. A busca local é interrompida quando não houver mais movimentos de melhora em relação à solução corrente, caracterizando um ótimo local com relação a esta vizinhança.

2.5.2 Busca Local BI_{TMM}

Esta busca local utiliza a vizinhança $N^TMM(.)$ através da estratégia *Best improvement*. Ela funciona como segue: Inicialmente, as máquinas são ordenadas, em ordem decrescente, de acordo com a função custo de cada máquina. A seguir, são analisados todas as trocas envolvendo as tarefas da máquina mais sobrecarregada. O vizinho $s' \in N^TMM(s)$ com o melhor valor da função custo é aceito se $f(s') < f(s)$. Enquanto houver melhora, o procedimento é repetido a partir da máquina mais sobrecarregada. Não havendo melhora na máquina atual, são analisados

os movimentos envolvendo as tarefas da segunda máquina mais sobrecarregada. Este procedimento é aplicado até um máximo de 30% da quantidade de máquinas. A exploração não envolve a totalidade das máquinas dado o elevado custo desta busca.

2.5.3 Procedimento $Perturb\_TMD$

Este procedimento utiliza a vizinhança $N^TMD(.)$. Ele funciona como segue: Inicialmente, as máquinas são ordenadas, em ordem decrescente, de acordo com a função custo de cada máquina. A seguir, são aplicadas perturbações envolvendo a máquina mais sobrecarregada com a máquina menos sobrecarregada. Uma perturbação $s' \in N^TMD(s)$ é aceita se houver redução no custo de uma das duas máquinas envolvidas. Observe que este procedimento pode gerar uma solução de qualidade inferior, ou seja, com piora no valor do *makespan*. Caso a perturbação seja aceita, o procedimento é interrompido; caso contrário, são analisadas todas as perturbações possíveis envolvendo a máquina mais sobrecarregada e a segunda menos sobrecarregada. O procedimento é interrompido quando uma perturbação for aceita ou se esgotarem as perturbações envolvendo tarefas de todos os pares de máquinas.

2.6 Procedimento $Perturb\_ILS$

Este procedimento consiste em perturbar a solução ótima local por meio de movimentos de inserção, de forma a explorar outras regiões do espaço de soluções. Ele funciona como segue: Inicialmente seleciona-se uma máquina de maneira aleatória. Depois, retira uma tarefa de forma aleatória desta máquina selecionada. A tarefa retirada será inserida em uma segunda máquina também escolhida de forma aleatória, na melhor posição desta segunda máquina. A melhor posição será aquela que acarretar o menor custo nesta segunda máquina selecionada.

Para controlar a quantidade de perturbações realizadas é utilizado um nível de perturbação l definido por $l + 1$ movimentos de inserção. O número máximo de níveis de perturbação permitidos é 3; assim ocorrerão 4 movimentos de inserção simultâneos, no máximo. O nível l de perturbação aumenta após $vezesNivel$ soluções perturbadas sem que haja melhoria na solução corrente. Por outro lado, quando é encontrada uma solução melhor, o nível de perturbação volta para o nível mais baixo ($l = 1$).

3 Resultados Computacionais

Para realizar os experimentos computacionais foram utilizados 360 problemas-teste do conjunto disponibilizado por [12] em [10]. Esses problemas-teste usados envolvem combinações de 50, 100 e 150 tarefas com 10, 15 e 20 máquinas, totalizando 9 grupos com 40 instâncias cada. Além das instâncias, em [10] são disponibilizados os melhores resultados encontrados para cada instância.

O algoritmo AIA foi desenvolvido em JAVA e os parâmetros adotados foram: *i*) $vezesNivel$ igual a 15; *ii*) $tempoExec = n \times (m/2) \times t$ milisegundos, sendo n o número total de tarefas, m o número total de máquinas e t um parâmetro testado para três valores: 10, 30 e 50. Esses valores de t são os mesmos adotados como critério de parada nos trabalhos de [12] e de [6].

O algoritmo AIA foi comparado com o melhor algoritmo desenvolvido por [12], denominado GA2, e com o algoritmo AIV de [6]. Para compará-los foi utilizada a métrica Desvio Percentual Relativo RPD, definida por $RPD_i = (\bar{f}_i^{Alg} - f_i^*)/(f_i^*)$. Nesta expressão, $\bar{f}_i^{Alg}$ é o valor da solução encontrada pelo algoritmo Alg para o problema-teste i, e f_i^* é a melhor solução conhecida.

Dado seu caráter estocástico, os algoritmos AIA e AIV foram executados 30 vezes para cada instância e para cada valor de t. Já em [12], o algoritmo GA2 foi executado 5 vezes para cada instância e para cada valor de t. A métrica usada para comparação entre os algoritmos é a Média do Desvio Percentual Relativo RPD_i^{avg} dos valores RPD_i encontrados.

Para cada conjunto de instâncias são encontrados três valores de RPD_i^{avg}, ambos separados por '/'. Esta separação representa os resultados dos testes para cada valor de t, na ordem $t = 10/30/50$. Os valores negativos indicam que os resultados alcançados pelos algoritmos superaram os melhores valores encontrados em [12] em seus experimentos.

Tabela 1: Média dos RPDs dos algoritmos AIA, AIV e GA2 para $t = 10/30/50$.

Instâncias	AIA[1]	AIV[1]	GA2[2]
50 x 10	**1.37/-0.45/-0.56**	3.69/1.83/1.30	7.79/6.92/6.49
50 x 15	**-2.05/-3.45/-3.99**	1.52/-0.77/-1.33	12.25/8.92/9.20
50 x 20	**-3.3/-4.6/-5.09**	5.26/2.01/1.65	11.08/8.04/9.57
100 x 10	**1.96/0.21/-0.41**	5.06/2.93/2.00	15.72/6.76/5.54
100 x 15	**-1.31/-3.06/-3.74**	1.80/-0.40/-1.29	22.15/8.36/7.32
100 x 20	**-3.67/-5.44/-6.03**	0.52/-1.64/-2.89	22.02/9.79/8.59
150 x 10	**1.59/-0.21/-0.9**	3.77/1.99/1.07	18.40/5.75/5.28
150 x 15	**-0.74/-2.76/-3.49**	1.83/-0.24/-1.04	24.89/8.09/6.80
150 x 20	**-4.03/-5.95/-6.59**	-1.04/-3.10/-4.00	22.63/9.53/7.40
RPD^{avg}	**-1.13/-2.86/-3.42**	2.49/0.29/-0.50	17.44/8.02/7.35

[1]Executados em um Intel Core i5 3.0 GHz, 8 GB de RAM, 30 execuções para cada instância
[2]Executados em um Intel Core 2 Duo 2.4 GHz, 2 GB de RAM, 5 execuções para cada instância

Os melhores valores de RPD^{avg} estão destacados em negrito. Analisando-os, verifica-se que o algoritmo AIA encontrou melhores resultados que os dos algoritmos AIV e GA2 em todos os grupos de instâncias. Uma tabela com todos os resultados obtidos por AIA juntamente com os melhores resultados da literatura dados por [12], são disponibilizadas em `http://www.decom.ufop.br/prof/marcone/projects/upmsp/Experiments_UPMSPST_AIA.ods`.

Pelos resultados completos é possível observar que, em relação aos resultados médios (RPD^{avg}) o algoritmo AIA supera os resultados da literatura, disponibilizados em [10], em 80,2% das instâncias, enquanto que em relação aos melhores resultados, o algoritmo AIA estabeleceu novos limites superiores em 95% das instâncias.

Para avaliar se existe diferença estatística entre os algoritmos através dos valores de RPD^{avg}, foi aplicada a análise de variância (ANOVA) [8], com 95% de confiança ($threshold = 0.05$), tendo-se encontrado $p = 2 \times 10^{-16}$. Como $p < threshold$, existem diferenças estatísticas entre eles. Para identificá-las, foi realizado o teste de Tukey HSD [8] com nível de confiança 95% e $threshold = 0.05$. Na Tabela 2 são apresentadas as diferenças nos valores de RPD^{avg} (diff), o limite inferior (lwr), o limite superior (upr) e o valor p para cada par de algoritmos.

Tabela 2: Resultados para o teste Tukey HSD.

Algoritmos	diff	lwr	upr	p
AIV-AIA	3.228889	0.6676046	5.790173	0.0096862
GA2-AIA	13.306296	10.7450120	15.867581	0.0000000
GA2-AIV	10.077407	7.5161231	12.638692	0.0000000

Pela Tabela 2 é possível observar que o algoritmo AIA se difere estatisticamente dos algoritmos AIV e GA2, porque os valores de p são menores que o $threshold$, assim como o algoritmo AIV também difere estatisticamente do algoritmo GA2.

4 Conclusões

Este trabalho tratou o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência (UPMSPST), tendo como objetivo a minimização do *makespan*. Para sua resolução foi proposto um algoritmo heurístico denominado AIA, que se baseia nos procedimentos heurísticos *Adaptive Shortest Processing Time* (ASPT), *Iterated Local Search* (ILS) e *Adaptive Local Search* (ALS). O procedimento ASPT é usado para gerar uma solução inicial. O ILS é usado para refinar essa solução inicial e utiliza o ALS como método de busca local. O procedimento ALS utiliza dois métodos de refinamento e um de perturbação. Esses métodos são aplicados de acordo com uma probabilidade, a qual depende do sucesso em aplicações anteriores.

O algoritmo AIA foi comparado com dois outros algoritmos da literatura, o GA2 de [12], e o AIV de [6], usando problemas-teste da literatura envolvendo até 150 tarefas e 20 máquinas. Pelos resultados encontrados foi possível mostrar a supremacia do algoritmo AIA, uma vez que

ele apresentou soluções finais de melhor qualidade e menor variabilidade em todos os grupos de instâncias. Essa supremacia foi comprovada estatisticamente.

Como trabalho futuro pretende-se incorporar um módulo de Programação Linear Inteira Mista (PLIM) para atuar como busca local na resolução de sub-problemas do UPMSPST.

Agradecimentos

Os autores agredecem às agências FAPEMIG, CNPq e CAPES, e à Universidade Federeal de Ouro Preto (UFOP), pelo apoio financeiro dado ao desenvolvimento deste trabalho.

Referências

[1] A. Al-Salem, Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times, *Engineering J. of the University of Qatar*, 17, No. 1 (2004) 177–187.

[2] J. Arnaout, G. Rabadi, R. Musa, A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times, *J. of Intelligent Manufacturing*, 21, No. 6 (2010) 693–701.

[3] K. R. Baker, "Introduction to Sequencing and Scheduling". John Wiley & Sons, 1974.

[4] M. Garey, D. Johnson, "Computers and intractability: A guide to the theory of np-completeness", WH Freeman & Co., San Francisco, 1979.

[5] R. Graham, E. Lawler, J. Lenstra, A. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, 5, No. 2 (1979) 287–326.

[6] M. N. Haddad, L. P. Cota, M. J. F. Souza, N. Maculan, AIV: A Heuristic Algorithm based on Iterated Local Search and Variable Neighborhood Descent for solving the Unrelated Parallel Machine Scheduling Problem with Setup Times, em "16th International Conference on Enterprise Information Systems", Lisboa, Portugal, 2014.

[7] H. R. Lourenço, O. Martin, T. Stützle, "Iterated Local Search," em Handbook of Metaheuristics, International Series in Operations Research & Management Science, F. Glover e G. Kochenberger (Eds). Kluwer Academic Publishers, Norwell, MA, 57 (2003) 321–353.

[8] D. Montgomery. "Design and Analysis of Experiments". John Wiley & Sons, New York, NY, 5ª edição, 2007.

[9] G. Rabadi, R. J. Moraga, A. Al-Salem, Heuristics for the unrelated parallel machine scheduling problem with setup times, *J. of Intelligent Manufacturing*, 17, No. 1 (2006) 85–97.

[10] SOA, 2011, sistemas de Optimización Aplicada. A web site that includes benchmark problem data sets and solutions for scheduling problems. Available at http://soa.iti.es/problem-instances.

[11] M. Souza, I. Coelho, S. Ribas, H. Santos, L. Merschmann, A hybrid heuristic algorithm for the open-pit-mining operational planning problem, *European J. of Operational Research*, 207, No. 2 (2010) 1041–1051.

[12] E. Vallada, R. Ruiz, A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times, *European J. of Operational Research*, 211, No. 3 (2011) 612–622.

[13] K.-C. Ying, Z.-J. Lee, S.-W. Lin, Makespan minimisation for scheduling unrelated parallel machines with setup times, *J. of Intelligent Manufacturing*, 23, No. 5 (2012) 1795–1803.

