CNMAC2012
XXXIV CONGRESSO NACIONAL DE
MATEMÁTICA APLICADA E COMPUTACIONAL
17 a 21 de setembro de 2012 – Águas de Lindóia/SP

ISSN 1984-8218

Algoritmos Heurísticos Híbridos para o Problema de Sequenciamento em Máquinas Paralelas Não-Relacionadas com Tempos de Preparação Dependentes da Sequência

Matheus N. Haddad, **Marcone J. F. Souza**,
Haroldo G. Santos, **Alexandre X. Martins**
Departamento de Computação, DECOM, UFOP
35400-000, Ouro Preto, MG
E-mails: mathaddad@gmail.com, marcone@iceb.ufop.br
haroldo.santos@gmail.com, xmartins@decea.ufop.br

Resumo: *Este trabalho trata o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência. Este problema é muito encontrado no âmbito industrial e pertence à classe NP-difícil. Visando a sua resolução, é proposto um algoritmo heurístico híbrido, nomeado GIVMP. Tal algoritmo combina as metaheurísticas Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS), Variable Neighborhood Descent (VND) e Path Relinking (PR) com um modelo de programação inteira mista (PIM) para resolver o Problema do Caixeiro Viajante Assimétrico (PCVA). A ideia do algoritmo é utilizar a fase de construção GRASP para gerar a solução inicial, o VND como responsável por realizar as buscas locais do ILS, o PR usado como estratégia de intensificação e diversificação da busca e o modelo PIM com a intenção de melhorar a melhor solução encontrada, realizando buscas locais na máquina gargalo. Para explorar o espaço de soluções são utilizados movimentos de inserção e troca de tarefas entre máquinas. O GIVMP foi testado em conjuntos de problemas-teste da literatura e seus resultados foram comparados com dois algoritmos genéticos da literatura. Os experimentos computacionais mostraram que os resultados alcançados pelo algoritmo proposto superaram os resultados da literatura, tanto em termos de qualidade da solução quanto na variabilidade das soluções. Também foram obtidos, para a maioria dos problemas-teste, novas melhores soluções.*

Palavras-chave: *Sequenciamento em máquinas paralelas, makespan, Iterated Local Search, Path Relinking, Variable Neighborhood Descent, Greedy Randomized Adaptive Search Procedure*

1 Introdução

Dentre os vários problemas de sequenciamento em máquinas paralelas (PMSP), destaca-se o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência (em inglês *Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times*), ou apenas UPMSP. Este problema se caracteriza por conter tempos de preparação entre as tarefas que são dependentes da máquina à qual as mesmas estão alocadas. Também existem tempos de preparação que são dependentes da sequência em que essas tarefas estão alocadas nas máquinas. O objetivo é minimizar o tempo máximo de conclusão do sequenciamento, o chamado *makespan*.

O UPMSP é um problema com importância tanto teórica quanto prática. Teórica, pois é um problema de difícil solução, por pertencer à classe NP-difícil e prática, pois há muitas situações em que ele aparece, como por exemplo, em processos de fabricações em indústrias têxteis [1].

ISSN 1984-8218

Em vista da dificuldade de encontrar soluções ótimas para o UPMSP em tempos computacionais aceitáveis em instâncias maiores, o problema tem sido resolvido na literatura por meio de métodos heurísticos. Em [2] é apresentada uma heurística de particionamento de três fases, chamada PH. Em [3] é implementada uma metaheurística para busca randômica priorizada. Essa metaheurística utiliza uma estratégia que combina heurísticas de construção e refinamento, utilizando ideias semelhantes ao *Greedy Ramdomized Adaptive Search Procedure* (GRASP). Em [4], os autores implementam o método de Colônia de Formigas (ACO) para estruturas especiais do problema, em que a proporção do número de tarefas e o número de máquinas é grande. [5] propõem um método restrito de *Simulated Annealing* (RSA) que reduz o esforço computacional da busca eliminando movimentos de tarefas que não serão efetivos. [6] apresentam uma hibridização do *Multi-start* com *Variable Neighborhood Descent* (VND) e programação matemática. O UPMSP é resolvido por meio de Algoritmos Genéticos em [7]. Nos dois algoritmos desses autores, nomeados GA1 e GA2, a população inicial é criada com um indivíduo pela heurística de múltipla inserção [8] e o resto da população é gerado aleatoriamente. Depois de criados, são aplicadas buscas locais em todos os indivíduos. A seleção para o cruzamento é realizada por torneio n-ário. Nos cruzamentos são realizados procedimentos de buscas locais limitadas. Buscas locais baseadas em movimentos de múltipla inserção entre máquinas também são aplicadas a todos os indivíduos. A seleção para a sobrevivência é feita pela estratégia estacionária, incluindo indivíduos originais na população, desde que sejam melhores que os piores. Os autores geraram problemas-teste para o UPMSP e disponibilizaram-os em [9].

Com o intuito de solucionar o UPMSP, é proposto um algoritmo heurístico híbrido, nomeado GIVMP. Este algoritmo tem como base o *Iterated Local Search* – ILS [10], porém com uma diferença para a maioria das aplicações de ILS, pois otimiza-se subpartes do problema a cada perturbação. A geração da solução inicial é feita por um procedimento de construção parcialmente gulosa, inspirada na fase construtiva do *Greedy Randomized Adaptive Search Procedure* – GRASP [11]. Na intenção de tornar mais eficiente a exploração do espaço de soluções, utilizou-se o método *Variable Neighborhood Descent* – VND [12], que é responsável por realizar as buscas locais do ILS. Além disso, também é aplicada a técnica *Path Relinking* – PR [13] com a estratégia *mixed relinking* com o objetivo de conectar ótimos locais gerados gerados a cada iteração do ILS com soluções elite formadas ao longo da busca. O algoritmo também inclui a utilização de um modelo de Programação Inteira Mista (PIM) que é aplicado ao final do ILS como estratégia de intensificação, buscando melhorar a melhor solução encontrada.

2 Caracterização do problema

No problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência (UPMSP) tem-se um conjunto $N = \{1, ..., n\}$ de n tarefas e um conjunto $M = \{1, ..., m\}$ de m máquinas, com as seguintes características: a) Cada tarefa deve ser processada exatamente uma vez por apenas uma máquina; b) Cada tarefa j possui um tempo de processamento p_{jk} que depende da máquina k na qual será alocada. Por esta característica, as máquinas são ditas não-relacionadas; c) Existem tempos de preparação entre as tarefas, s_{ijk}, em que k representa a máquina cujas tarefas i e j serão processadas, nesta ordem. Tais tempos de preparação são dependentes da sequência e da máquina. A dependência da sequência é mostrada pela desigualdade $s_{ijk} \neq s_{jik}$ e a dependência da máquina pode ser percebida nesta desigualdade $s_{ijk} \neq s_{ijh}$, onde h é uma outra máquina; d) Considera-se, também, o tempo de preparação para processar a primeira tarefa, representado por s_{0jk}.

O objetivo é encontrar um sequenciamento das n tarefas nas m máquinas de forma a minimizar o tempo máximo de conclusão do sequenciamento, o chamado de *makespan*, ou $C_{\max}$. Pelas características citadas, o UPMSP é definido como $R_M \mid S_{ijk} \mid C_{\max}$ [14].

O PMSP com máquinas idênticas e sem tempos de preparação ($P_M \mid\mid C_{\max}$) pertence à classe NP-difícil, mesmo quando se tem duas máquinas [15]. Como o UPMSP é uma generalização do $P_M \mid\mid C_{\max}$, logo ele também pertence à classe NP-difícil.

ISSN 1984-8218

3 Metodologia

Uma solução s do UPMSP é representada como um vetor de listas. Em tal representação se tem um vetor v cujo tamanho é o número de máquinas, m, e cada posição desse vetor contém um número que representa uma máquina. O sequenciamento das tarefas em cada máquina, por sua vez, é representado por uma lista de números, em que cada número representa uma tarefa. O valor de avaliação de cada solução é o *makespan*.

O algoritmo proposto, nomeado GIVMP, é um algoritmo heurístico híbrido que combina as metaheurísticas *Greedy Randomized Adaptive Search Procedure* (GRASP), *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) e *Path Relinking* (PR) com um modelo de programação inteira mista (PIM) para resolver o Problema do Caixeiro Viajante Assimétrico (PCVA). A ideia do algoritmo é utilizar a fase de construção GRASP para gerar a solução inicial, o VND como responsável por realizar as buscas locais do ILS, o PR usado como estratégia de intensificação e diversificação da busca e o modelo PIM com a intenção de melhorar a melhor solução encontrada. O pseudocódigo deste algoritmo é mostrado no Algoritmo 1.

Algoritmo 1: GIVMP

```
   Entrada: vezesnivel, tempoExec
 1 Timer tempo.start();
 2 Solucao s, s', melhorSol;
 3 elite ← {};
 4 s .geraSolucaoGRASP(0.2);
 5 s .VNDAleatorio() ;
 6 atualizaMelhor(s,melhorSol);
 7 elite .insere(s);
 8 nivel ← 1;
 9 enquanto tempo.tempoAtual() ≤ tempoExec × 0.95 faça
10     s' ← s;
11     vezes ← 0;
12     perturbmax ← nivel + 1;
13     enquanto vezes < vezesnivel faça
14         perturb ← 0;
15         s' ← s;
16         enquanto perturb < perturbmax faça
17             perturb ++;
18             s'.perturbacao() ;
19         fim
20         s'.avalia();
21         s'.VNDAleatorio() ;
22         elite.atualiza(s');
23         pr ← aleatorio(0,1);
24         se pr ≤ 0.1 ∧ elite.tamanho ≥ 5 então
25             el ← aleatorio(1,5);
26             MxPR(elite [el], s') ;
27         fim
28         se s'.fo < s.fo então
29             s ← s';
30             atualizaMelhor(s,melhorSol);
31             elite.atualiza(s);
32         fim
33         vezes ++;
34     fim
35     nivel ++;
36     se nivel ≥ 4 então
37         nivel ← 1 ;
38     fim
39 fim
40 melhorSol ← buscaLocalPIM(melhorSol, tempoExec × 0.05) ;
41 Retorne melhorSol;
```

O Algoritmo 1 recebe como entrada o número de vezes em cada nível de perturbação, *vezesnivel*, e o tempo limite de execução do algoritmo, dado por *tempoExec*. No início de sua execução há a inicialização de três soluções (linha 2), e também ocorre a criação do conjunto de soluções elite (linha 3). Em seguida, na linha 4, é gerada uma solução parcialmente gulosa por meio da fase construtiva GRASP. Tal solução passa por processos de buscas locais na linha 5, por meio do módulo VND. Com o resultado destas buscas locais, a melhor solução conhecida até então, *melhorSol*, é atualizada. A seguir, a solução criada é inserida no conjunto elite (linhas 6-7).

O processo iterativo do GIVMP está situado nas linhas 9 a 39 e termina quando o tempo de execução for maior que o tempo limite, *tempoExec*. Uma cópia da solução atual é feita na linha 10. O laço seguinte (linhas 13-34) é responsável por controlar o número de vezes em cada nível de perturbação, *vezesnivel*. O próximo laço, linhas 16 a 19, executa as perturbações, sendo que o número de vezes que o laço é executado depende do nível de perturbação. Com as perturbações realizadas, a solução obtida é avaliada na linha 20. Em seguida, é aplicado o módulo VND nesta

ISSN 1984-8218

solução e é verificado se o ótimo local alcançado, em relação a todas vizinhanças adotadas no VND, pode ser inserido no conjunto elite (linhas 21 e 22). As linhas 23 a 27 aplicam a técnica PR, utilizando a estratégia *mixed relinking*, com uma probabilidade de 10% e quando o conjunto elite estiver completo, com cinco soluções. Nas linhas 28 a 32 é averiguado se as alterações feitas na solução contribuíram para uma melhor qualidade da mesma. O processo iterativo termina quando o tempo de execução é maior que 95% de *tempoExec*. Ao final do processo iterativo (linha 40), é aplicado o modelo PIM na melhor solução encontrada (*melhorSol*), restringindo seu tempo de execução em 5% do *tempoExec*.

A solução inicial é gerada de forma parcialmente gulosa, seguindo a ideia da fase de construção da metaheurística GRASP. Esta construção usa como guia uma função de avaliação g, que avalia para cada máquina seu tempo de término. Dada uma lista de tarefas candidatas (LC), é avaliada a inserção de cada uma delas em todas as máquinas por meio da função g. A seguir, é construída uma Lista Restrita de Candidatos (LRC) das tarefas de LC melhor classificadas. Integram a LRC as tarefas j tais que $g(j) \leq g_{\min} + \alpha \times (g_{\max} - g_{\min})$, sendo $g_{\min}$ o menor tempo de conclusão produzido pelas tarefas de LC, $g_{\max}$ o maior tempo de conclusão e α um parâmetro responsável por controlar o quão gulosa será a solução. A seguir, uma tarefa j é escolhida aleatoriamente de LRC e alocada à máquina respectiva. O processo é repetido até que todas as tarefas sejam alocadas.

O módulo *VNDAleatorio*, linhas 5 e 21 do Algoritmo 1, usa o método VND, do inglês *Variable Neighborhood Descent* [12], com três procedimentos de busca local escolhidos em uma ordem aleatória. No primeiro, são aplicados movimentos de múltipla inserção, caracterizados por retirar uma tarefa de uma máquina e inserí-la em uma posição de outra máquina, seguindo uma estratégia *First Improvement*. O segundo procedimento de busca trabalha com movimentos de trocas de tarefas entre máquinas diferentes, enquanto o terceiro explora o espaço de soluções com movimentos de trocas de tarefas pertencentes a uma mesma máquina. O método é encerrado quando é encontrado um ótimo local com relação aos três tipos de movimentos considerados.

As perturbações consistem na aplicação de movimentos de inserção em um ótimo local. Cada perturbação se caracteriza por retirar uma tarefa de uma máquina qualquer e inserí-la em outra máquina qualquer. Ao inserir a tarefa em outra máquina, procura-se sua melhor posição de inserção, ou seja, uma posição em que a máquina tenha o menor tempo de conclusão. Desta forma, subpartes do problema são otimizadas a cada perturbação. A quantidade de modificações em uma solução é controlada por um "nível" de perturbação, de forma que um determinado nível k de perturbação consiste na aplicação de $k+1$ movimentos de inserção. O nível máximo permitido é 3; assim, ocorrerão 4 perturbações simultâneas, no máximo. O objetivo ao aumentar o nível de perturbação é diversificar a busca e procurar por uma solução de melhora em uma região gradativamente mais "distante" daquele ótimo local. O nível de perturbação somente é aumentado após geradas *vezesnivel* soluções perturbadas sem que haja melhoria da solução corrente. Por outro lado, sempre que uma melhor solução é encontrada, a perturbação volta ao seu nível mais baixo.

O módulo *Path Relinking* (PR) é aplicado entre uma solução proveniente da busca local e uma solução do conjunto elite. Compõem o conjunto elite soluções que tenham *makespan* menores que o da melhor solução já presente no mesmo ou soluções melhores que a pior solução do conjunto, mas com um nível mínimo de diversidade das outras soluções elite. O tamanho do conjunto elite está limitado a 5 soluções e o nível mínimo de diversidade entre as soluções é de 10%. A diversidade entre duas soluções para o UPMSP é definida como o percentual de tarefas diferentes nas mesmas posições.

Na linha 40 do Algoritmo 1, é acionado o *solver* CPLEX 12.1 com a formulação de programação inteira mista (PIM) relativa ao Problema do Caixeiro Viajante Assimétrico (PCVA). Resolver o PCVA é equivalente a resolver o problema de encontrar a melhor sequência das tarefas em uma máquina no UPMSP. Esta chamada é feita à máquina gargalo, isto é, àquela cujo tempo de término é o maior dentre todas as máquinas. Se a nova sequência encontrada tiver o tempo de conclusão menor que o da sequência anterior, então esta nova sequência é atribuída

ISSN 1984-8218

à máquina. A seguir, o *makespan* é atualizado. Se ainda houver tempo para mais uma busca, primeiro é verificado se a máquina gargalo foi alterada; caso isso aconteça, é realizada uma nova busca local, aplicando-se a formulação PIM nessa máquina. Se no entanto ela continua a mesma, são realizadas chamadas iterativas do módulo *VND* aplicado a s^*, seguido do módulo *PR*, sendo este último aplicado a alguma solução do conjunto elite e a s^*. Assim que o tempo se esgotar, é retornada a solução s^* tendo as sequências de suas máquinas otimizadas.

A formulação PIM utilizada, dada pelas equações 1-8, foi a de [16]. Neste modelo, as variáveis binárias de decisão x_{ij} assumem valor 1 se a cidade i precede imediatamente a cidade j em um circuito e valor 0, caso contrário. Além disso, também são definidas as variáveis binárias de decisão y_{ij} que recebem valor 1 se a cidade i precede, não necessariamente imediatamente, a cidade j em um circuito e 0, caso contrário.

$$min \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij} \tag{1}$$

$$s.a.$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \qquad \forall j = 1, .., n \tag{2}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1 \qquad \forall i = 1, .., n \tag{3}$$

$$y_{ij} \geq x_{ij} \qquad \forall i, j = 2, .., n, i \neq j \tag{4}$$

$$y_{ij} + y_{ji} = 1 \qquad \forall i, j = 2, .., n, i \neq j \tag{5}$$

$$y_{ij} + y_{jk} + y_{ki} \leq 2 \qquad \forall i, j, k = 2, .., n, i \neq j \neq k \tag{6}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\ i, j = 1, .., n, i \neq j \tag{7}$$

$$y_{ij} = 0 \qquad \forall\ i, j = 2, .., n, i \neq j \tag{8}$$

4 Resultados Computacionais

Para a realização dos testes computacionais, foi utilizado um conjunto de 360 problemas-teste da literatura, encontrados em [9], envolvendo combinações de 50, 100 e 150 tarefas e 10, 15 e 20 máquinas. Para cada combinação dessas existem 40 instâncias. Em [9] também são fornecidos os melhores valores de *makespan* conhecidos até então nesses problemas-teste.

O GIVMP foi implementado na linguagem C++. Todos os experimentos foram executados em um computador com processador *Intel Core 2 Quad 2,4 GHz* com 4 GB de memória RAM e sistema operacional *Ubuntu 10.10*. Os parâmetros utilizados foram: $vezesnivel = 15$ e $\alpha = 0,2$. Como critério de parada, *tempoExec*, considerou-se o tempo máximo de duração do processamento $Tempo_{\max}$, em milissegundos, calculado pela Eq. (9), sendo m o número de máquinas, n o número de tarefas e t o tempo, em milisegundos. Tal como em [7], para t foram usados três valores para cada instância: 10, 30 e 50.

$$Tempo_{\max} = n \times (m/2) \times t \quad ms \tag{9}$$

Na métrica utilizada para a comparação dos algoritmos, dada pela Eq. (10), o objetivo é verificar a variabilidade das soluções finais produzidas pelos algoritmos. Nesta medida, calcula-se, para cada algoritmo *Alg* aplicado a um problema-teste i, o desvio percentual relativo RPD_i da solução encontrada $\bar{f}_i^{Alg}$ em relação à melhor solução f_i^* conhecida até então. O algoritmo GIVMP foi executado 30 vezes, para cada instância e para cada valor de t, ao final dessas 30 execuções calcula-se, para cada instância, a média RPD_{avg} dos valores de RPD_i encontrados.

$$RPD_i = \frac{\bar{f}_i^{Alg} - f_i^*}{f_i^*} \tag{10}$$

A Tabela 1 mostra, para os conjuntos de instâncias, o RPD_i^{avg} para cada valor de t do algoritmo GIVMP, bem como dos algoritmos GA1 e GA2 de [7]. Destaca-se que o GA1 e o GA2 foram executados em máquinas com uma configurações semelhantes às utilizadas para os testes, possibilitando a comparação. Para cada conjunto de instâncias são encontrados três valores de RPD_i^{avg} separados. Esta separação representa resultados de testes alterando os valores de t. Valores negativos indicam que os resultados encontrados superam os melhores resultados

ISSN 1984-8218

encontrados por [7]. Valores destacados em negrito indicam os melhores valores médios de RPD. Como se observa, o GIVMP obteve os melhores resultados, pois além de ganhar em todas as instâncias, ainda superou a maioria dos melhores resultados de [7].

Tabela 1: Médias dos Desvios Percentuais Relativos (RPD_{avg}) com $t = 10, 30, 50$

	GA1[1]			GA2[1]			GIVMP		
Instâncias	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**	**t=10**	**t=30**	**t=50**
50x10	13,56%	12,31%	11,66%	7,79%	6,92%	6,49%	**1,79%**	**0,58%**	**-0,11%**
50x15	13,87%	13,95%	12,74%	12,25%	8,92%	9,20%	**-0,39%**	**-2,10%**	**-2,57%**
50x20	12,92%	12,58%	13,44%	11,08%	8,04%	9,57%	**3,92%**	**1,43%**	**-0,05%**
100x10	13,11%	10,46%	9,68%	15,72%	6,76%	5,54%	**2,66%**	**0,99%**	**0,42%**
100x15	15,41%	13,95%	12,94%	22,15%	8,36%	7,32%	**-0,11%**	**-1,88%**	**-2,74%**
100x20	15,34%	13,65%	13,60%	22,02%	9,79%	8,59%	**-1,66%**	**-3,65%**	**-4,60%**
150x10	10,95%	8,19%	7,69%	18,40%	5,75%	5,28%	**2,11%**	**0,14%**	**-0,70%**
150x15	14,51%	11,93%	11,78%	24,89%	8,09%	6,80%	**1,08%**	**-1,03%**	**-1,78%**
150x20	13,82%	12,66%	12,49%	22,63%	9,53%	7,40%	**-1,91%**	**-3,90%**	**-4,75%**
Média Total	13,72%	12,19%	11,78%	17,44%	8,02%	7,35%	**0,83%**	**-1,05%**	**-1,87%**

[1]Testes realizados em um PC Intel Core 2 duo com 2.4 GHz e 2 GB de RAM, 5 iterações para cada instância

De forma a verificar se existe diferença estatística entre os algoritmos em questão (GA1, GA2 e GIVMP), foi realizada uma análise de variância (ANOVA), que resultou em um valor $p = 2,2 \times 10^{-16}$. Como p é menor que o $threshold = 0,05$, significa que existe diferença estatística entre os algoritmos.

Com a intenção de verificar onde estão essas diferenças, foi realizado um teste de Tukey HSD, com nível de confiança de 95% e $threshold = 0,05$. A Tabela 2 contém as diferenças nos valores médios de RPD (diff), o limite inferior (lwr), o limite superior (upr) e o valor p (p) para cada par de algoritmos. Pode ser visto pelos valores de p que o GIVMP se difere estatisticamente tanto do GA1 quanto do GA2, por conta desses valores serem menores que o *threshold*. Por outro lado, os algoritmos genéticos, GA1 e GA2, não são diferentes estatisticamente.

Tabela 2: Resultados do teste Tukey HSD

Algoritmos	diff	lwr	upr	p
GA2-GA1	-1,626296	-4,083793	0,8312008	0,2598164
GIVMP-GA1	-13,259259	-15,716756	-10,8017622	0,0000000
GIVMP-GA2	-11,632963	-14,090460	-9,1754659	0,0000000

Agradecimentos

Os autores agradecem à FAPEMIG, CNPq e CAPES pelo apoio na execução deste trabalho.

5 Conclusões

Este trabalho abordou o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência, tendo por objetivo a minimização do *makespan*. Para resolvê-lo, foi proposto um algoritmo heurístico híbrido, nomeado GIVMP, que combina os procedimentos heurísticos GRASP, ILS, VND e *Path Relinking* com um modelo de programação inteira mista para resolver o Problema do Caixeiro Viajante Assimétrico (PCVA).

Utilizando conjuntos de problemas-teste da literatura, o algoritmo proposto foi comparado com dois algoritmos genéticos da literatura. Os resultados computacionais mostraram a capacidade do GIVMP em produzir soluções de qualidade para o UPMSP, com menores variabilidades e conseguindo superar as melhores soluções conhecidas. Também foi feita uma análise de variância nos resultados produzidos pelos algoritmos, tendo-se verificado a diferença estatística entre eles. Desta forma, pode-se afirmar que o GIVMP é o melhor algoritmo nas instâncias usadas.

Referências

[1] M. J. Pereira Lopes and J. M. de Carvalho. A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research*, 176:1508–1527, 2007.

ISSN 1984-8218

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

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

[4] J.P. Arnaout, G. Rabadi, and R. Musa. 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, 2010.

[5] Kuo-Ching Ying, Zne-Jung Lee, and Shih-Wei Lin. Makespan minimisation for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, 2010. doi:10.1007/s10845-010-0483-3.

[6] K. Fleszar, C. Charalambous, and K.S. Hindi. A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, 2011. doi:10.1007/s10845-011-0522-8.

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

[8] M. Kurz and R. Askin. Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39:3747–3769, 2001.

[9] SOA, 2011. Problemas-teste para o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência. Disponíveis em `http://soa.iti.es/problem-instances`.

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

[11] T. Feo and M. Resende. Greedy randomized search procedure. *Journal of Global Optimization*, 6:109–133, 1995.

[12] Nenad Mladenovic and Pierre Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[13] F. Glover. Tabu search and adaptive memory programming - advances, applications and challenges. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, 1996.

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

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

[16] S.C. Sarin, H.D. Sherali, and A. Bhootra. New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. *Operations research letters*, 33(1):62–70, 2005.