GRASP com *Path Relinking* Aplicado na Resolução do Problema de Programação de Tarefas em Máquinas Paralelas com *Setup Times* Dependentes da Seqüência e de Recursos

Edmar H. Kampke, José E. C. Arroyo, André G. Santos

Departamento de Informática, Universidade Federal de Viçosa
Avenida P. H. Rolfs s/n, Campus UFV, 36570-000 Viçosa, MG, Brasil
edmar.kampke@ufv.br, jarroyo@dpi.ufv.br, andre@dpi.ufv.br

Abstract. This work deals with the parallel machine scheduling problem with resource-assignable sequence dependent setup times. The goal of the problem is to minimize the total completion time and the total assignes resources. Due to the combinatorial complexity of this problem, an algorithm based on heuristic GRASP is proposed, in which the randomness parameter used in the construction phase is self-adjusted, according to the quality of the solutions previously found (Reactive GRASP). The algorithm uses an intensification strategy based on the Path Relinking technique which consists in exploring paths between elite solutions found by the Reactive GRASP. The obtained results are compared with the best results available in literature and show the good performance of the proposed algorithm.

Keywords: Metaheuristics, Task Scheduling, Setup Time, GRASP, Path Relinking.

1 Introdução

Neste trabalho é abordado um problema de Programação de Tarefas em Máquinas Paralelas (PTMP), no qual são considerados tempos de preparação (*setup times*) de máquinas, que dependem de recursos disponíveis (por exemplo, mão-de-obra). Um dos primeiros trabalhos propostos para o problema PTMP estudou o problema considerando a minimização de *setups times* [11]. Posteriormente, o problema foi modelado e métodos heurísticos foram aplicados para minimizar outros critérios, tais como: média dos tempos de conclusão (*completions times*) das tarefas, média dos atrasos das tarefas e o tempo máximo de conclusão (*makespan*) [8].

Nos trabalhos sobre problemas de PTMP, nota-se uma tendência de aplicação de metaheurísticas, tais como Busca Tabu [3][6], *Simulated Annealing* [9][13] e GRASP [1][5].

O problema de PTMP considerado neste trabalho consiste em determinar o melhor seqüenciamento de n tarefas em um conjunto de m máquinas paralelas diferentes (uma tarefa é processada em uma única máquina). Cada tarefa j possui um tempo de processamento na máquina i (p_{ij}) e se a tarefa k é processada logo após a tarefa j na

máquina i, existe um tempo de preparação S_{ijk} cuja duração depende da quantidade de recurso disponível R_{ijk}. Cada tempo S_{ijk} pode variar entre dois valores S_{ijk}^- (*setup time* mínimo) e S_{ijk}^+ (*setup time* máximo). Similarmente, R_{ijk} pode variar entre R_{ijk}^- (recurso mínimo) e R_{ijk}^+ (recurso máximo). Os *setup times* e os recursos são relacionados de forma linear: se é utilizado um número mínimo (máximo) de recursos então a duração do *setup time* será o maior (menor) possível. Dessa forma, o problema de PTMP abordado neste trabalho, relaciona *setup time* e recursos, e é aqui denotado por PTMPSR.

O objetivo do problema é minimizar simultaneamente dois critérios: tempo total de conclusão das tarefas e número de recursos utilizados na preparação das máquinas. O tempo de conclusão (*completion time*) da tarefa j na máquina i é denotado por C_{ij}. A função objetivo deste problema é definida como:

$$Z = \lambda \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{\substack{k=1 \\ k \neq j}}^{n} R_{ijk} + \delta \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij}$$

onde, λ e δ representam as importâncias ou pesos atribuídos aos critérios e $R_{ijk} = 0$ se a tarefa j não for precedente da tarefa k na máquina i. Similarmente, se a tarefa j não for processada na máquina i, então $C_{ij} = 0$ [14].

O problema PTMPSR descrito acima é provado ser NP-Difícil [14]. Esta prova consiste em simplesmente reduzir o problema a um clássico problema de PTMP com *setup times* dependentes da seqüência, que é NP-Difícil [15].

A Figura 1 ilustra um exemplo de solução para um caso do problema, com $n = 4$ tarefas e $m = 2$ máquinas. As tarefas 4 e 2 são processadas, nesta ordem, na máquina 1, finalizando nos tempos $C_{14} = 43$ e $C_{12} = 155$, respectivamente. O *setup time* entre estas tarefas é $S_{142} = 61$, sendo utilizados $R_{142} = 3$ unidades de recursos. Da mesma forma, as tarefas 3 e 1 são processadas na máquina 2 e são finalizadas nos tempos $C_{23} = 27$ e $C_{21} = 100$, respectivamente. O *setup time* entre estas tarefas é $S_{231} = 28$ com utilização de $R_{231} = 4$ unidades de recursos.

Neste trabalho, uma solução do problema é representada por um arranjo de tamanho $n+m-1$ contendo as n tarefas e $m-1$ elementos (-1) utilizados para dividir as tarefas em m grupos, um para cada máquina. A solução mostrada na Figura 1 é representada pelo arranjo [4, 2, -1, 3, 1]. Note que os grupos de tarefas [4,2] e [3,1] são processadas respectivamente nas máquinas 1 e 2.

Foram propostas, na literatura, três heurísticas construtivas para resolver o problema PTMPSR [14]. A diferença entre as heurísticas está no critério de ordenação das tarefas na construção da solução. As heurísticas são denominadas *Shortest Processing Time with Setups Resource Assignment* (SPTSA), *Shortest Processing and Setup Time with Setups Resource Assignment* (SPSTSA) e *Dynamic Job Assignment with Setups Resource Assignment* (DJASA). A heurística SPTSA ordena as tarefas em ordem crescente pelo menor tempo de processamento da tarefa em todas as máquinas. A heurística SPSTSA é similar a SPTSA, a única diferença é que a média dos *setup times* é considerada junto com o menor tempo de processamento. A heurística DJASA constrói a solução de forma dinâmica, ordenando crescentemente as tarefas que ainda não foram incluídas na solução parcial, pelo incremento que elas proporcionarão na função objetivo se forem incluídas naquele momento. Ruiz e Andrés [14] verificaram que a heurística DJASA apresenta os melhores resultados.

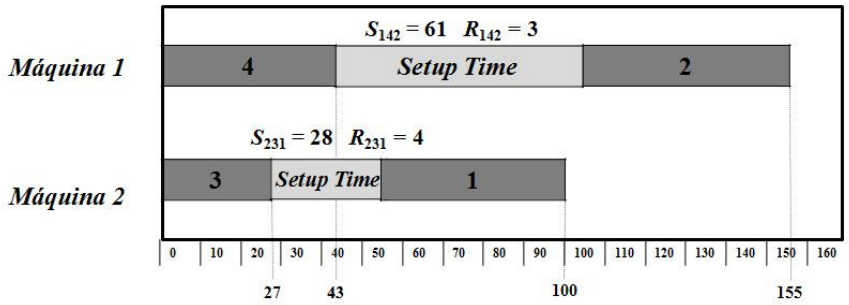

Fig. 1. Exemplo de solução.

Este artigo é organizado da seguinte maneira. Na Seção 2 são apresentados os detalhes da implementação do algoritmo proposto. Os testes realizados e resultados computacionais são mostrados na Seção 3. A Seção 4 apresenta as conclusões do trabalho.

2 GRASP com *Path Relinking* para o problema PTMPSR

Para a resolução do problema PTMPSR, é proposto um algoritmo que utiliza a metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP), proposta inicialmente por Feo e Resende [4], acrescida de um mecanismo reativo [12] para a definição do parâmetro de aleatoriedade utilizado na construção da solução inicial. O algoritmo utiliza também a técnica *path relinking* como estratégia de intensificação, que consiste em explorar trajetórias que conectam soluções de alta qualidade.

```
Procedimento GRASP (α, CritérioParada)
1       f_min ← +∞;
2       enquanto não CritérioParada faça
3          s_1 ← Construção_Solução(α);
4          s_2 ← Busca_Local(s_1);
5          se f(s_2) < f_min então
6             s ← s_2;
7             f_min ← f(s_2);
8          fim-se
9       fim-enquanto;
10      retorne s;
fim GRASP;
```

Fig. 2. Pseudocódigo genérico da heuristica GRASP.

A metaheurística GRASP é um método de múltiplas partidas, na qual cada iteração consiste de duas fases: uma fase de construção de uma solução viável e uma fase de busca local, na qual se procura melhorar a qualidade da solução construída na fase

anterior. A melhor solução encontrada em todas as iterações da metaheurística é retornada como resultado. Os únicos parâmetros a serem definidos na metaheurística GRASP são o parâmetro de aleatoriedade α e o critério de parada, que geralmente é o número de iterações da metaheurística. Na Figura 2, apresenta-se um pseudocódigo genérico da metaheurística GRASP. Esta metaheurística tem como entrada o parâmetro α, que é utilizado na fase de construção de soluções, e o critério de parada. Para valores pequenos de α, as soluções são construídas com um maior grau de "gulosidade" e para valores maiores de α, as soluções são construídas de forma mais aleatória. Da Figura 2, nota-se que a metaheurística GRASP iterativamente constrói uma solução s_1 (passo 3) e esta solução é melhorada por um procedimento de busca local (passo 4). Sempre é armazenada a melhor solução encontrada até o momento (passos 5 a 7).

2.1 Construção de Soluções

Na fase de construção, uma solução (seqüência de tarefas) para o problema PTMPSR é gerada iterativamente. Iniciando com uma seqüência vazia, a cada iteração, é adicionada uma única tarefa à seqüência parcial. Para escolher a tarefa a ser adicionada, é definida uma Lista de Candidatos (LC) com todas as tarefas ainda não seqüenciadas. Para cada tarefa dessa lista são feitas simulações de inclusão da tarefa nas máquinas da seqüencia parcial. As tarefas desta lista são então ordenadas de forma crescente pelo incremento que elas proporcionarão ao valor da função objetivo. Esta forma de ordenação das tarefas também é usada na heurística construtiva DJASA [14]. Na heurística DJASA sempre é adicionada a primeira tarefa, ou seja, a tarefa que apresenta o menor incremento ao valor da função objetivo. No algoritmo de construção implementado neste trabalho, ao invés de escolher a primeira tarefa, é escolhida aleatoriamente uma tarefa, dentre as η primeiras tarefas da lista LC. A tarefa escolhida é adicionada na máquina da seqüência parcial que proporciona o menor incremento na função objetivo. As η primeiras tarefas da LC formam uma Lista Restrita de Candidatas (LRC) cujo tamanho depende do parâmetro α. O valor de η é definida como MAX(1,α×|LC|). A Figura 3 mostra o algoritmo de construção de soluções. O algoritmo finaliza quando a seqüencia contiver todas as n tarefas.

```
Procedimento Construção_Solução (α)
1       LC ← {t1,...,tn};        // Lista com todas as tarefas
2       Seqüência ← { };
3       enquanto ( | Seqüência | < n ) faça
4          Ordena_LC( );
5          η ← MAX(1, α × |LC|);
6          i ← Random(1, η);
7          Seqüência ← Seqüência ∪ {ti};
8          LC ← LC - {ti};
9       fim-enquanto;
10      retorne Seqüência;
fim Construção_Solução;
```

Fig. 3. Algoritmo de construção de soluções.

2.2 Busca Local

A Busca Local é um procedimento iterativo que consiste em melhorar uma solução s_1 procurando novas soluções vizinhas dela. Estas soluções vizinhas são obtidas realizando algumas alterações (movimentos) na estrutura da solução atual s_1. Escolhe-se um vizinho s dentre todos os vizinhos de s_1. Se o vizinho escolhido s é melhor que s_1, a busca continua a partir de s (ou seja, $s_1 \leftarrow s$). O procedimento finaliza quando não é possível melhorar a solução atual s_1, ou seja, quando s_1 é um ótimo local. Neste trabalho foram utilizados movimentos de inserção para a obtenção de soluções vizinhas. Um vizinho de s_1 é gerado inserindo uma tarefa que está na posição i da seqüência em outra posição j, tal que $1 \leq i, j \leq n$ e $i \neq j$. Por exemplo, para a seqüência [4, 2, -1, 3, 1] da Figura 1, as seqüências vizinhas [4, -1, 3, 2, 1] e [2, 4, -1, 3, 1] são geradas através da inserção da tarefa 2 após a tarefa 3 e antes da tarefa 4, respectivamente. Realizando este tipo de movimento é possível gerar $k(k\text{-}1)$ soluções vizinhas, onde $k = n+m\text{-}1$ (tamanho da seqüência).

2.3 Ajuste do parâmetro α (GRASP Reativo)

Para a resolução do problema PTMPSR, abordado neste trabalho, é proposta a aplicação da heurística GRASP Reativo[12], denotada neste trabalho por GR, que tem como principal característica o auto-ajuste do parâmetro de aleatoriedade α, utilizado na fase de construção, de acordo com a qualidade das soluções encontradas nas iterações anteriores.

A heurística GR funciona da seguinte maneira: define-se um conjunto A de v possíveis valores para o parâmetro α, $A = \{\alpha_1, \alpha_2, ..., \alpha_v\}$. Inicialmente, todos os valores $\alpha_k \in A$ tem a mesma probabilidade de serem escolhidos, ou seja, $p_k = 1/v$. A cada iteração escolhe-se um valor $\alpha_k \in A$ (com probabilidade p_k) para o parâmetro α. As probabilidades p_k são recalculadas a cada γ iterações. Os valores de α_k que produzem melhores soluções terão maior probabilidade p_k, conseqüentemente, nas próximas iterações terão maior chance de serem escolhidos.

De acordo com a estrutura da heurística GR, a quantidade de soluções construídas utilizando α_k é armazenada num arranjo *count* e a soma dos valores da função objetivo dessas soluções é armazenada num arranjo *score*. Considera-se f_{min} o menor valor da função objetivo encontrada até o momento. Os valores de p_k serão atualizados, a cada γ iterações, utilizando *count*, *score*, f_{min} e um parâmetro de amplificação θ.

Na Figura 4 é apresentado o pseudocódigo da heurística GR, que utiliza a estratégia de Reconexão de Caminhos, apresentada na próxima seção. Nota-se que nos passos 3 a 7, são inicializadas as variáveis e estruturas usadas no ajuste do parâmetro α. No passo 9 seleciona-se aleatoriamente, com probabilidade p_k, um $\alpha_k \in A$ que será utilizado na iteração atual. No passo 24 é incrementado o número de soluções construídas com α_k e é acumulado o valor da função objetivo da solução encontrada nessa iteração. Os passos 26 a 28, correspondentes à atualização das probabilidades p_k, são executados a cada γ iterações.

Na implementação da heurística GR foi considerado o seguinte conjunto de valores para o parâmetro α: $A = \{0,1;0,2;...;0,9\}$. Os valores dos parâmetros γ e θ foram calibrados e os melhores resultados foram obtidos para $\gamma = 20$ e $\theta = 10$.

2.4 Intensificação com *Path Relinking*

A Reconexão de Caminhos (*Path Relinking*), proposta por Glover [7], é uma estratégia de intensificação e diversificação das soluções obtidas na fase de busca local. Dado um par de soluções, denominadas origem e guia, o objetivo desta estratégia é construir um caminho de soluções entre a solução origem e a solução guia, por meio da troca das tarefas na seqüência da solução origem. No contexto do GRASP, o *path relinking* foi introduzido pela primeira vez por Laguna e Martí [10].

Neste trabalho o *path relinking* mantém um conjunto de soluções elite E [2]. O procedimento tem início com a escolha aleatória da solução guia $s_g \in E$, desde que $s_g \neq s_s$, onde s_s denota a solução origem obtida pela busca local. As soluções de E e a solução origem s_s são ótimos locais. O *path relinking* tenta descobrir soluções melhores que s_s e s_g, que ainda não foram avaliadas.

Em cada etapa analisa-se todas as possíveis trocas que incorporam atributos da solução guia s_g na solução origem s_s. É escolhida a troca que melhora mais (ou deteriora menos) a solução origem. A solução s_s obtida após a movimentação é armazenada se melhorar a melhor solução atual. O procedimento continua, iterativamente, até que s_s e s_g se tornem iguais.

Para uma solução origem que possui k elementos em posições diferentes da solução guia, são necessárias $k-1$ movimentações de troca até que as soluções sejam idênticas. Na exploração desse caminho, $k-2$ soluções vizinhas, diferentes da solução origem e da solução guia, são avaliadas.

No algoritmo proposto, o *path relinking* está acoplado à heurística GRASP Reativo (GR), sendo aplicado como refinamento de ótimos locais. Durante a execução do GR, as soluções de alta qualidade são armazenadas no conjunto E. A heurística tem como entrada o parâmetro E_{Tam}, que é o tamanho máximo do conjunto E, e o critério de parada.

O conjunto E não permite a inclusão de uma solução repetida. Na execução do GRASP reativo, enquanto $|E|$ for menor que E_{Tam}, a solução obtida pela fase de busca local é inserida em E, desde que seja diferente de todas as soluções contidas em E. Caso $|E| = E_{Tam}$, então comparamos a solução obtida após a aplicação do *path relinking* com a pior solução do conjunto E, atualizando o conjunto se for o caso.

Dessa forma, na Figura 4 apresenta-se a estrutura genérica da heurística GRASP Reativo com *Path Relinking*, denotado por GRPR. O passo 2 demonstra que o conjunto de soluções elite é inicializado como vazio. No passo 12 verifica-se o conjunto E já foi totalmente preenchido. No passo 13 escolhe-se aleatoriamente a solução s_2 em E. Em seguida aplica-se o *path relinking* para construir um caminho de soluções partindo da solução s_1 (obtida pela busca local no passo 11) para a solução s_2. A solução retornada pelo *path relinking* é armazenada em s_3 (passo 14). No passo 15 aplica-se o *path relinking* novamente. Dessa vez para construir um caminho de soluções partindo de s_2 para s_1. A solução retornada é armazenada em s_4. No passo 16 a melhor das soluções s_3 e s_4, retornadas pelo *path relinking*, é armazenada em s. Caso s não faça parte do conjunto elite e seja diferente de s_1, aplica-se a busca local em s (passo 18). No passo 20 o conjunto E é atualizado e nos passos 31 e 32 a melhor solução de E é retornada como solução do método.

Na implementação das heurísticas GR e GRPR, adota-se como critério de parada o tempo computacional, baseado no número de tarefas (n) e número de máquinas (m) de

cada problema. O tempo computacional utilizado como critério de parada é de $(n \times m)/2$ segundos. Por exemplo, no problema com $n = 100$ tarefas e $m = 20$ máquinas o tempo de execução é 1000 segundos. Na heurística GRPR o tamanho do conjunto de soluções elite foi definido por $E_{Tam} = 10$.

Procedimento *GRPR* (E_{Tam}, *CritérioParada*)
1 $f_{min} \leftarrow +\infty$;
2 $E = \emptyset$;
3 $iter \leftarrow 1$;
4 Defina $A = \{\alpha_1, \alpha_2, ..., \alpha_v\}$
5 **para** $k \leftarrow 1$ *até* v **faça**
6 $count[k] \leftarrow 0$; $score[k] \leftarrow 0$; $p_k \leftarrow 1/v$;
7 **fim-para;**
8 **enquanto não** *CritérioParada* **faça**
9 $\alpha \leftarrow$ Selecione $\alpha_k \in A$ com probabilidade de escolha p_k;
10 $s \leftarrow$ Construção_Solução (α);
11 $s_1 \leftarrow$ Busca_Local(s);
12 **se** $|E| = E_{Tam}$ **então**
13 $s_2 \leftarrow$ Escolha aleatoriamente uma solução elite de E;
14 $s_3 \leftarrow$ Path_Relinking(s_1, s_2);
15 $s_4 \leftarrow$ Path_Relinking(s_2, s_1);
16 $s \leftarrow$ Melhor_Solucao(s_3, s_4);
17 **se** $s \notin E$ **e** $s \neq s_1$ **então**
18 $s \leftarrow$ Busca_Local(s);
19 **fim-se**
20 ATUALIZA_E(s);
21 **senão**
22 $E \leftarrow E \cup \{s_1\}$;
23 **fim-se**
24 $count[k] \leftarrow count[k] + 1$; $score[k] \leftarrow score[k] + f(s_2)$;
25 **se** $iter \mod \gamma = 0$ **então**
26 $avg[k] \leftarrow score[k]/count[k]$ para todo $k \in \{1,2,...,v\}$;
27 $\sigma \leftarrow \Sigma\, (f_{min}/avg[k])^{\theta}$ para todo $k \in \{1,2,...,v\}$;
28 $p_k \leftarrow (f_{min}/avg[k])^{\theta}/\sigma$ para todo $k \in \{1,2,...,v\}$;
29 **fim-se**
30 **fim-enquanto;**
31 $s \leftarrow$ Melhor_Solucao(E);
32 retorne s;
fim *GRPR*;

Fig. 4. Estrutura Genérica da Heurística GRPR.

3 Resultados Computacionais

Neste artigo, testa-se o desempenho das heurísticas GRASP Reativo sem *Path Relinking* (GR) e GRASP Reativo com *Path Relinking* (GRPR), na resolução do problema PTMPSR. Para tal são utilizados os problemas teste gerados por Ruiz e Andrés [14], que são disponibilizados em *http://www.upv.es/gio/rruiz*. Eles geraram um total de 720 problemas, divididos em dois grupos: *small* e *large* (cada um com 360 problemas). O grupo *small* contém problemas com número de tarefas $n \in \{6,8,10\}$ e número de máquinas $m \in \{3,4,5\}$. Já no grupo de problemas *large* os valores para o número de tarefas (n) e máquinas (m) pertencem respectivamente aos seguintes conjuntos: {50,75,100} e {10,15,20}.

Em todos os problemas, os parâmetros λ e δ, que representam os pesos dos critérios otimizados, foram definidos como: λ=50 e δ=1 [14].

As heurísticas GR e GRPR foram implementadas na linguagem de programação Java, com a versão 1.6, e executadas usando o compilador JDK 6.0. Os problemas teste foram resolvidos utilizando um computador com processador Intel® Core™ Quad com 2.4GHz e 3GB de memória RAM.

3.1 Comparação dos resultados obtidos

Os resultados obtidos pelas heurísticas são comparados com os melhores resultados disponibilizados na literatura[14]. As soluções disponibilizadas para o grupo de problemas *small* foram obtidas pelo software CPLEX (versão 9.1) através da resolução do Modelo Matemático de Programação Inteira (MMPI) [14]. A execução do CPLEX foi limitada a 300 segundos para problemas com n = 6 tarefas e 3600 segundos para problemas com n = 8 e n = 10 tarefas. Para todos os problemas com n = 6 e n = 8, o CPLEX determinou a solução ótima. Nos problemas com n = 10 somente foram obtidas soluções aproximadas, devido ao limite do tempo de execução do CPLEX.

As soluções disponibilizadas para o grupo de problemas *large* são os melhores resultados obtidos pelas heurísticas construtivas propostas por Ruiz e Andrés [14]. Na Tabela 1, as colunas "*Média GR*" e "*Média GRPR*" exibem, respectivamente, as médias dos resultados encontrados pelas heurísticas GR e GRPR, para problemas que possuem o número de tarefas (n) e número de máquinas (m) iguais. As colunas "CPLEX" e "*Heurísticas Construtivas*" exibem as médias dos resultados disponibilizados na literatura para os problemas dos grupos *small* e *large*, respectivamente.

Para os problemas do grupo *small*, os resultados obtidos pelas heurísticas GR e GRPR foram bastante satisfatórios. Nos 120 problemas com n = 6 tarefas, as heurísticas GR e GRPR encontraram a solução ótima em 114 e 117 problemas, respectivamente. Em todos os 120 problemas com n = 8 tarefas, ambas as heurísticas encontraram a solução ótima. Nos 120 problemas restantes, com n = 10 tarefas, ambas as heurísticas, GR e GRPR, quando comparadas ao CPLEX, encontraram soluções superiores em 90 problemas e soluções idênticas em 30 problemas. Para o grupo *small* de problemas, a porcentagem média de melhoria das soluções obtidas pelas

heurísticas GR e GRPR, em relação aos resultados do CPLEX foi de 1,15% em ambas as heurísticas.

Tabela 1. Comparação de resultados das heurísticas GR e GRPR com os melhores resultados da literatura [14].

Problemas *small*				**Problemas large**			
n*×*m	**Média do CPLEX**	**Média GR**	**Média GRPR**	***n*×*m***	**Média dos Melhores Resultados**	**Média GR**	**Média GRPR**
6 × 3	**738**	739	**738**	50 × 10	12.828	11.721	**11.684**
6 × 4	474	474	474	50 × 15	8.656	7.982	**7.964**
6 × 5	287	288	288	50 × 20	6.195	5.741	**5.732**
8 × 3	1.353	1.353	1.353	75 × 10	26.099	23.489	**23.469**
8 × 4	943	943	943	75 × 15	18.290	**16.945**	16.963
8 × 5	683	683	683	75 × 20	14.014	13.016	**12.997**
10 × 3	2.116	**2.076**	**2.076**	100 × 10	41.517	37.962	**37.921**
10 × 4	1.503	**1.477**	**1.477**	100 × 15	30.316	28.008	**28.007**
10 × 5	1.117	1.077	**1.076**	100 × 20	23.708	22.287	**22.268**
Média	1.023,7	1.022,2	1.022,0	**Média**	20.180,3	18.572,3	18.556,1

Para os 360 problemas do grupo *large*, as heurísticas GR e GRPR obtiveram, em todos os casos, resultados superiores em relação aos melhores resultados heurísticos disponibilizados na literatura. Observa-se pela Tabela 1 que dos 9 grupos de problemas *large* que possuem as mesmas dimensões *n*×*m*, em 8 deles a heurística GRPR superou os resultados obtidos pela heurística GR. Comparando os resultados obtidos pelas heurísticas GR e GRPR, para cada um dos 360 problemas do grupo *large*, foi observado que em 197 problemas o resultado da heurística GRPR superou a heurística GR. Em 3 problemas os resultados obtidos por ambas as heurísticas foram idênticos e a heurística GR mostrou resultado superior, com relação a heurística GRPR, nos 160 problemas restantes. No grupo *large* a porcentagem média de melhoria das soluções obtidas pelas heurísticas GR e GRPR, em relação aos melhores resultados disponibilizados, foram de 7,97% e 8,05%, respectivamente.

4 Conclusão

Neste trabalho foi proposta uma heurística GRASP Reativo (GR) na qual é testada o comportamento da técnica *Path Relinking* (GRPR) para resolver o problema de programação de tarefas em máquinas paralelas, denominado PTMPSR. O problema PTMPSR foi recentemente formulado na literatura e até o momento não existem trabalhos de aplicação da heurística GRASP. O desempenho da heurística proposta foi testado em 720 problemas, de porte variado. As soluções obtidas pela heurística foram comparadas com as melhores soluções encontradas na literatura. Para os problemas de grande porte, a heurística GRPR obteve uma melhoria média de 8,05% com relação às melhores soluções heurísticas disponibilizadas na literatura, o que demonstra a eficácia da heuristica GRASP. A heurística GRPR, em média, obteve desempenho superior à heurística GR para problemas de pequeno e grande porte, o

que demonstra um ganho na qualidade das soluções com a utilização do *path pelinking*. Como trabalho futuro sugere-se o desenvolvimento de métodos híbridos que combinem a Heurística GRASP com *Simulated Annealing* ou *Iterated Local Search* (ILS).

Agradecimentos. Este trabalho foi financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq e pela Fundação de Amparo à Pesquisa do Estado de Minas Gerais – FAPEMIG.

Referências

1. Aiex, R.M., Binato, S. e Resende, M.G.C. (2003), Parallel grasp with path-relinking for job shop scheduling. Parallel Comput., 29(4), 393–430.
2. Aiex, R.M., Resende, M.G.C., Pardalos, P.M. e Toraldo, G. (2005), Grasp with path relinking for three-index assignment. INFORMS Journal on Computing, 17(2), 224–247.
3. Armentano, V.A. e Yamashita, D.S. (2000), Tabu search for scheduling on identical parallel machines to minimize mean tardiness. Journal of Intelligent Manufacturing, 11,453–460.
4. Feo, T.A. e Resende, M. (1989), A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, 8, 67–71.
5. Feo, T.A., Sarathy, K. e McGahan, J. (1996), A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties. Comput. Oper. Res., 23(9), 881–895.
6. Franca, P.M., Gendreau, M., Laporte, G. e Muller, F.M. (1996), A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. International Journal of Production Economics, 43(2–3), 79–89.
7. Glover, F. (1996), Tabu search and adaptive memory programming - advances, applications and challenges. Em: Barr, R.S., Helgason, R.V. and Kenngington, J.L. editors. Interfaces in Computer Science and Operations Research, 1–75.
8. Guinet, A. (1990), Textile production systems: A succession of non-identical parallel processor shops. Journal of the Operational Research Society,42, 655–671.
9. Kim, D.W., Kim, K.H., Jang, W. e Chen, F.F. (2002), Unrelated parallel machine scheduling with setup times using simulated annealing. Robotics and Computer Integrated Manufacturing, 18, 223–231.
10. Laguna, M. e Marti, R. (1999), Grasp and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing, 11(1), 44–52.
11. Marsh, J.D. e Montgomery, D.C. (1973), Optimal procedures for scheduling jobs with sequence dependent changeover times on parallel processors. AIIE Technical Papersg, 279–286.
12. Prais, M. e Ribeiro, C.C. (2000), Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. INFORMS Journal on Computing, 12(3), 164–176.
13. Radhakrishna, S. e Ventura, J.A. (2000), Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. International Journal of Production Research, 38(10), 2233–2252.
14. Ruiz, R. e Andrés, C. (2007), Unrelated parallel machines scheduling with resource-assignable sequence dependent setup times. Em Baptiste, P., Kendall, G., Munier-Kordon, A., Sourd, F., eds.: Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Paris, France, 439–446.
15. Webster, S.T. (1997), The complexity of scheduling job families about a common date. Operations Research Letters, 20(2), 65–74.