GRASP Reativo e Iterated Local Search aplicado ao problema de programação de tarefas em máquinas paralelas com tempos de preparação dependentes da sequência e de recursos

Edmar Hell Kampke[1], José Elias Cláudio Arroyo[2], André Gustavo dos Santos[2]

[1]Departamento de Engenharia Rural – Universidade Federal do Espírito Santo – UFES
Alegre – ES

[2]Departamento de Informática – Universidade Federal de Viçosa – UFV
Viçosa – MG

mazinhok@yahoo.com.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, it is proposed algorithms based on metaheuristics Reactive GRASP and Iterated Local Search (ILS). The obtained results for each metaheuristic are compared with the best results available in literature. The results show the good performance of the proposed algorithms.*

Resumo. *Este trabalho aborda o problema de sequenciamento de tarefas em máquinas paralelas, com tempos de preparação das máquinas dependente da sequência e do número de recursos utilizados. O objetivo do problema é minimizar o tempo total de conclusão das tarefas e número total de recursos utilizados na preparação das máquinas. Dada a complexidade combinatória do problema, propõem-se algoritmos baseados nas metaheurísticas GRASP Reativo e Iterated Local Search (ILS). Os resultados obtidos em cada metaheurística são comparados entre si e com os melhores resultados disponíveis na literatura. Os resultados apresentam o bom desempenho dos algoritmos propostos.*

1. Introdução

As modernas indústrias de manufatura têm sido estimuladas a tornar seus processos de produção mais eficientes, principalmente por causa da competitividade crescente imposta pelas transformações que têm afetado a ordem econômica mundial. As principais finalidades da programação da produção são: administrar os recursos de uma empresa industrial, sequenciar de forma eficiente os pedidos nos centros de trabalho e atender aos prazos de entrega dos produtos vendidos. Os problemas de programação de tarefas em máquinas encontram-se intimamente ligados ao planejamento e programação da produção, e são muito estudados na literatura científica.

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 foi apresentado por Marsh e Montgomery (1973). Eles estudaram o problema considerando a minimização de *setups times*. Guinet (1991) modelou matematicamente o problema e aplicou métodos heurísticos 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*).

Nos trabalhos mais recentes sobre problemas de PTMP, nota-se uma tendência de aplicação de metaheurísticas, tais como *Simulated Annealing* (Kim *et al.,*2002), GRASP (Laguna e González-Velarde, 1991; Armentano e França Filho, 2007) e *Iterated Local Search* (Tang e Luo, 2006).

2. Descrição do Problema

O problema considerado neste trabalho consiste em determinar o melhor sequenciamento 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 Problema de Programação de Tarefas em Máquinas Paralelas com Setup time e Recursos (PPTMPSR), e foi abordado pela primeira vez por Ruiz e Andrés (2007).

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}. Conforme foi proposto por Ruiz e Andrés (2007), 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 $C_{ij} = 0$.

O PPTMPSR descrito acima é provado ser NP-Difícil (Ruiz e Andrés, 2007). Esta prova consiste em simplesmente reduzir o problema a um clássico problema de PTMP com *setup times* dependentes da sequência, que é NP-Difícil (Webster, 1997).

O PPTMPSR é de suma importância para as indústrias de manufatura. Este problema ocorre, por exemplo, na fabricação de cerâmicas, onde existem diferentes

máquinas de polimento. Cada máquina inicia o polimento de um tipo de cerâmica, e ao final a máquina precisa ser limpa e preparada para realizar o polimento de outro tipo de cerâmica. Este tempo de limpeza e preparação varia de acordo com a máquina, com o tipo da cerâmica polida anteriormente, com o tipo de cerâmica que será polida em seguida e com o número de recursos utilizados, que neste caso, pode ser, por exemplo, o número de pessoas que realizarão a limpeza e ajuste da máquina.

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.

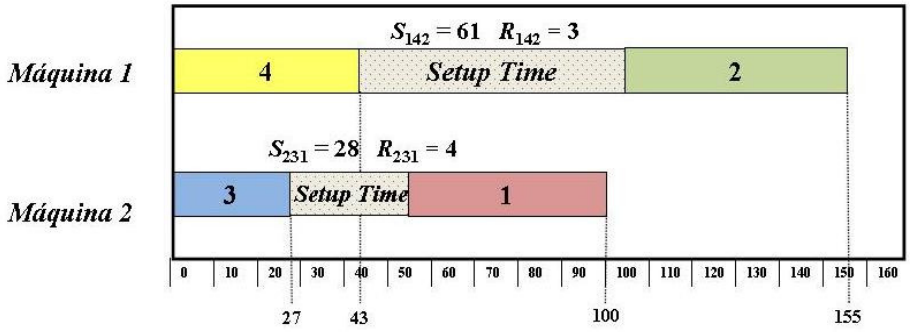

Figura 1. Exemplo de solução.

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.

Ruiz e Andrés (2007) propuseram três heurísticas construtivas para resolver o PPTMPSR. A diferença entre as heurísticas está no critério de ordenação das tarefas, utilizado na construção de uma 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). Esta última 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 (2007) verificaram que a heurística DJASA apresenta os melhores resultados.

Neste artigo são propostos dois algoritmos, um deles utiliza a metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) acrescida de um mecanismo reativo para a definição do parâmetro de aleatoriedade utilizado na construção da solução inicial. O outro é baseado na metaheurística ILS (*Iterated Local Search*). Conforme dito anteriormente, há uma tendência em utilizar tais metaheurísticas para a solução de problemas de PTMP, pois em grande parte dos casos, apresentam resultados

satisfatórios. Para testar os algoritmos, utiliza-se 720 problemas disponíveis na literatura. Os resultados obtidos em cada metaheurística são comparados entre si e com os melhores resultados encontrados pelas heurísticas de Ruiz e Andrés (2007).

3. GRASP Reativo

Para a resolução do PPTMPSR, é proposto um algoritmo que utiliza a metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP), proposta inicialmente por Feo e Resende (1989), acrescida de um mecanismo reativo (Prais e Ribeiro, 2000) para a definição do parâmetro de aleatoriedade utilizado na construção da solução inicial.

```
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;
```

Figura 2. Pseudocódigo Genérico da Metaheurística GRASP.

A metaheurística GRASP é um método iterative 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. 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).

3.1. Construção de soluções

Na fase de construção, uma solução (sequência de tarefas) para o PPTMPSR é gerada iterativamente. Iniciando com uma sequência vazia, a cada iteração, é adicionada uma única tarefa à sequência parcial. Para escolher a tarefa a ser adicionada, é definida uma Lista de Candidatos (LC) com todas as tarefas ainda não sequenciadas. Para cada tarefa dessa lista são feitas simulações de inclusão da tarefa nas máquinas da sequencia 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 (Ruiz e Andrés, 2007). 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 sequência 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 sequencia contiver todas as *n* tarefas.

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

Figura 3. Algoritmo de Construção de Soluções.

3.2. Busca Local

A Busca Local também 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 sequência em outra posição j, tal que $1 \leq i, j \leq n$ e $i \neq j$. Por exemplo, para a sequência [4, 2, -1, 3, 1] da Figura 1, as sequências 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 sequência).

3.3. Ajuste do Parâmetro α

A metaheurística GRASP Reativo (Prais e Ribeiro, 2000), denotada neste trabalho por GR, 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.

```
Procedimento GR (CritérioParada)
1    f_min ← +∞;
2    iter ← 1;
3    Defina A = {α_1, α_2,..., α_v}
4    para k ← 1 até v faça
5      count[k] ← 0; score[k] ← 0; p_k ← 1/v;
6    fim-para;
7    enquanto não CritérioParada faça
8      α ← Selecione α_k ∈A com probabilidade de escolha p_k;
9      s_1 ← Construção_Solução(α);
10     s_2 ← Busca_Local(s_1);
11     se f(s_2) < f_min então
12       s ← s_2;
13       f_min ← f(s_2);
14     fim-se
15     count[k] ← count[k] + 1; score[k] ← score[k] + f(s_2);
16     se iter mod γ = 0 então
17       avg[k] ← score[k]/count[k] para todo k ∈{1,2,...,v};
18       σ ← Σ (f_min/avg[k])^Θ para todo k ∈{1,2,...,v};
19       p_k ← (f_min/avg[k])^Θ/σ para todo k ∈{1,2,...,v};
20     fim-se
21     iter ← iter + 1;
22   fim-enquanto;
23   retorne s;
fim GR;
```

Figura 4. Estrutura Genérica da Metaheurística GR.

A metaheurística 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, consequentemente, nas próximas iterações terão maior chance de serem escolhidos.

De acordo com a estrutura da metaheurística 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 metaheurística GR. Nota-se que nos passos 2 a 6, são inicializadas as variáveis e estruturas usadas no ajuste do parâmetro α. No passo 8 seleciona-se aleatoriamente, com probabilidade p_k, um $\alpha_k \in A$ que será utilizado na iteração atual. No passo 15 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 17 a 19, 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 $\theta = 10$.

3.4. Critério de Parada

Neste trabalho 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. Note que, quanto maior os valores de n e m, maior será o tempo de execução do algoritmo.

4. Iterated Local Search (ILS)

Iterated Local Search (ILS) é um algoritmo metaheurístico, proposto por Lourenço *et al.,* (2002), baseado na idéia de que um procedimento de busca local pode ser melhorado, gerando-se novas soluções de partida, as quais são obtidas por meio de perturbações numa solução ótima local. A perturbação deve permitir que a busca local explore diferentes soluções e, além disso, deve evitar um reinício aleatório. O método ILS é, portanto, um método de busca local que procura focar a busca não no espaço completo de soluções, mas em um pequeno subespaço, definido por soluções que são ótimas locais (Lourenço *et al.,* 2002).

```
Procedimento ILS (CritérioParada, d, T)
1    s1 ← Construção_Solução_Inicial;
2    s ← s1;
3    enquanto não CritérioParada faça
4       s2 ← Perturbação(s1, d);
5       s2 ← Busca_Local(s2);
6       Criterio_Aceitação(s, s1, s2, T);
7    fim-enquanto;
8    retorne s;
fim ILS;
```

Figura 5. Pseudocódigo Genérico da Metaheurística ILS.

Na Figura 5, apresenta-se um pseudocódigo genérico da metaheurística ILS, que possui 4 etapas principais: Obtenção de uma solução ótima local inicial s_1 (passo 1), perturbação da solução s_1 obtendo uma solução s_2 (passo 4), melhoria da solução s_2 (busca local - passo 5) e um critério de aceitação da solução atual (passo 6). As três

últimas etapas são executadas iterativamente enquanto o critério de parada não seja atendido. O algoritmo retorna a melhor solução obtida durante toda sua a execução. A metaheurística ILS é fortemente dependente da solução inicial ótima local. Na etapa de construção de soluções é utilizada a heurística gulosa construtiva DJASA, já detalhada em seções anteriores, seguida do procedimento de busca local.

Neste algoritmo, o procedimento de busca local e o critério de parada serão os mesmos já descritos para o algoritmo anterior. Os parâmetros a serem definidos na metaheurística ILS, além do critério de parada, são a quantidade d de elementos perturbados e o pârametro T utilizado no critério de aceitação.

4.1. Perturbação

O procedimento *Perturbação* modifica a solução corrente, por meio de movimentos aleatórios de remoção de tarefas na solução corrente. Esses movimentos são responsáveis por alterar a solução corrente, guiando-a para uma solução intermediária. Dessa forma, o mecanismo de perturbação deve ser forte o suficiente para permitir escapar do ótimo local corrente e permitir também a exploração de diferentes regiões do espaço de soluções. Ao mesmo tempo, a perturbação precisa ser fraca o suficiente para guardar características do ótimo local corrente, evitando o reinício aleatório.

Neste trabalho, o procedimento *Perturbação* é composto de 2 etapas: *Destruição* e *Construção*. Na primeira são escolhidas, de forma aleatória e sem repetição, d tarefas que serão removidas da solução corrente s_1. O movimento de remoção das d tarefas cria duas subsequências, a primeira de tamanho d, denotada por s_R, contém as tarefas removidas. A segunda subsequência, de tamanho $k - d$, onde $k = n+m$-1 (tamanho da sequência), é a sequência original s_1 sem as tarefas removidas.

```
Procedimento Perturbação(s1, d)
1    sR ← ∅
2    enquanto ( |sR| < d ) faça
3        ti ← Escolhe aleatoriamente uma tarefa de s1;
4        sR ← sR ∪ {ti};              //Insere a tarefa selecionada em sR
5        s1 ← s1 - {ti};              //Remove a tarefa selecionada de s1
6    fim-enquanto;
7    enquanto ( |sR| > 0) faça
8        s1 ← s1 ∪ sR[1];      //Insere a primeira tarefa de sR na melhor posição de s1
9        sR ← sR - sR[1];      //Remove a primeira tarefa de sR
10   fim-enquanto;
11   retorne s1;
fim Perturbação;
```

Figura 6. Pseudocódigo do procedimento de Perturbação.

Após a criação das duas subsequências, é aplicada a etapa de *Construção*. Nesta etapa, iterativamente, todas as tarefas de s_R são reinseridas na solução original s_1. A cada iteração, a primeira tarefa de s_R é removida e inserida em todas as possíveis posições de s_1. Cada sequência obtida é avaliada, e no final a que possui o menor valor da função

objetivo é considerada para a próxima iteração. O procedimento encerra-se quando todas as tarefas de s_R forem reinseridas em s_1.

Na Figura 6 apresenta-se um pseudocódigo genérico do procedimento de *Perturbação*. No passo 1 a sequência s_R é inicializada vazia. A etapa iterativa de *Destruição* é apresentada nos passos 3 a 5. Os passos 8 e 9 mostram a etapa iterativa de *Construção*, onde em cada iteração a primeira tarefa de s_R é removida e inserida em s_1, na posição em que apresentará o menor valor da função objetivo.

4.2. Critério de Aceitação

O *Critério de Aceitação* define se a solução ótima local retornada pela busca local será aceita ou descartada. Uma solução é aceita se ela melhorar a melhor solução atual. No algoritmo ILS, soluções que pioram a melhor solução atual podem também ser aceitas com uma pequena probabilidade. Dessa forma, evita-se a utilização constante da melhor solução atual e, consequentemente, uma rápida estagnação das soluções avaliadas. Neste trabalho utiliza-se um critério de aceitação similar ao usado pela metaheurística *Simulated Annealing*. No entanto, o valor do parâmetro *Temperatura*, neste caso é constante:

$$Temperatur\ a = T \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}{n \times m \times 10}$$

O valor da *Temperatura* depende dos tempos de processamento (p_{ij}) das tarefas, do número total de tarefas n, do número de máquinas m e do parâmetro T. Esse critério de aceitação é usado nos trabalhos de Osman e Potts (1989) e Ruiz e Stutzle (2008). Neste critério de aceitação a solução s_2 obtida após a busca local, é comparada com a solução atual s_1 que foi submetida à *Pertubação*. Caso s_2 seja melhor que s_1, a solução s_2 é aceita ($s_1 \leftarrow s_2$) para ser perturbada nas próximas iterações. Se s_2 melhorar a melhor solução atual s, então a solução s também é atualizada ($s \leftarrow s_2$). No caso de s_2 ser pior que s_1, s_2 poderá ser aceita para ser perturbada nas iterações seguintes, se:

$$r < exp\{-(f(s_2) - f(s_1))/Temperatura\}$$

onde r é um número aleatório no intervalo [0, 1] e *exp* é a função exponencial.

Neste trabalho, os parâmetros d e T, utilizados no procedimento de perturbação e no critério de aceitação, respectivamente, foram calibrados e os melhores resultados foram obtidos para $d = 4$ ($6 \leq n \leq 10$), $d = 10$ ($n > 10$) e $T = 0{,}5$.

5. Resultados Computacionais

Neste artigo, testa-se o desempenho das metaheurísticas GRASP Reativo(GR) e *Iterated Local Search* (ILS), na resolução do PPTMPSR. Para tal são utilizados os problemas teste gerados por Ruiz e Andrés (2007), 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 (Ruiz e Andrés, 2007).

As metaheurísticas GR e ILS 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.

5.1. Comparação dos Resultados Obtidos

Os resultados obtidos pelas metaheurísticas são comparados com os melhores resultados disponibilizados na literatura (Ruiz e Andrés, 2007). 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) (Ruiz e Andrés, 2007). 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 (2007). Na Tabela 1, as colunas "*Média GR*" e "*Média ILS*" exibem, respectivamente, as médias dos resultados encontrados pelas metaheurísticas GR e ILS, para a função objetivo (*Z*), nos problemas que possuem o mesmo número de tarefas (*n*) e o mesmo número de máquinas (*m*). As colunas "CPLEX" e "*Heurísticas Construtivas*" exibem as médias dos resultados disponibilizados na literatura para os problemas dos grupos *small* e *large*, respectivamente.

Tabela 1. Comparação de resultados das heurísticas GR e ILS com os melhores resultados da literatura (Ruiz e Andrés, 2007).

Problemas *small*				Problemas large			
n*×*m	**Média do CPLEX**	**Média GR**	**Média ILS**	***n*×*m***	**Média dos Melhores Resultados**	**Média GR**	**Média ILS**
6 × 3	**738**	739	**738**	50 × 10	12.828	11.721	**11.674**
6 × 4	474	474	474	50 × 15	8.656	7.982	**7.901**
6 × 5	**287**	288	**287**	50 × 20	6.195	5.741	**5.689**
8 × 3	**1.353**	**1.353**	1.354	75 × 10	26.099	23.489	**23.236**
8 × 4	**943**	**943**	949	75 × 15	18.290	16.945	**16.826**
8 × 5	**683**	**683**	684	75 × 20	14.014	13.016	**12.855**
10 × 3	2.116	**2.076**	2.080	100 × 10	41.517	37.962	**37.488**
10 × 4	1.503	**1.477**	1.484	100 × 15	30.316	28.008	**27.707**
10 × 5	1.117	**1.077**	1.079	100 × 20	23.708	22.287	**22.063**
Média	1.023,7	**1.012,2**	1.014,3	**Média**	20.180,3	18.572,3	**18.382,1**

Para os problemas do grupo *small*, os resultados obtidos pelas metaheurísticas GR e ILS foram bastante satisfatórios. Nos 120 problemas com $n = 6$ tarefas, as metaheurísticas GR e ILS encontraram a solução ótima em 114 e 120 problemas, respectivamente. Em todos os 120 problemas com $n = 8$ tarefas, a metaheurística GR encontrou a solução ótima, enquanto que a metaheurística ILS encontrou a solução ótima em 102 problemas. Nos 120 problemas restantes, com $n = 10$ tarefas, as metaheurísticas, GR e ILS, quando comparadas ao CPLEX, encontraram soluções superiores em 90 e 86 problemas e soluções idênticas em 30 e 34 problemas, respectivamente. Para o grupo *small* de problemas, a porcentagem média de melhoria das soluções obtidas pelas metaheurísticas GR e ILS, em relação aos resultados do CPLEX, foi de 1,12% e 0,92%, respectivamente.

Para os 360 problemas do grupo *large*, as metaheurísticas GR e ILS 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 $n \times m$, em todos a metaheurística ILS superou os resultados obtidos pela metaheurística GR. Comparando os resultados obtidos pelas metaheurísticas GR e ILS, para cada um dos 360 problemas do grupo *large*, foi observado que em 297 problemas o resultado da metaheurística ILS superou a heurística GR. Em 4 problemas os resultados obtidos por ambas as heurísticas foram idênticos e a metaheurística GR mostrou resultado superior, com relação a metaheurística ILS, nos 59 problemas restantes. No grupo *large* a porcentagem média de melhoria das soluções obtidas pelas metaheurísticas GR e ILS, em relação aos melhores resultados disponibilizados, foram de 7,97% e 8,91%, respectivamente.

6. Conclusões

Neste trabalho foram propostas duas metaheurísticas: GRASP Reativo (GR) e *Iterated Local Search* (ILS) para resolver o problema de programação de tarefas em máquinas paralelas, denominado PPTMPSR. O PPTMPSR foi recentemente formulado na literatura e até o momento não existem trabalhos de aplicação e comparação de resultados com diferentes metaheurísticas. O desempenho das metaheurísticas propostas foram testados em 720 problemas, de porte variado. As soluções obtidas pelas metaheurísticas foram comparadas com as melhores soluções encontradas na literatura. Para os problemas de grande porte, a metaheurística ILS obteve uma melhoria média de 8,91% com relação às melhores soluções heurísticas disponibilizadas na literatura, o que demonstra a eficácia desta metaheurística. A metaheurística GR, em média, obteve desempenho superior à metaheurística ILS para problemas de pequeno porte. Como trabalho futuro sugere-se o desenvolvimento de métodos híbridos que combinem as metaheurísticas GRASP Reativo com *Iterated Local Search* (ILS), ou até mesmo com outras metaheurísticas, como exemplo: *Simulated Annealing* e Algoritmos Genéticos.

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.

References

Armentano, V.A. e França Filho, M.F. (2007), "Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory based GRASP approach". European Journal of Operational Research, 183, 100–114.

Feo, T.A. e Resende, M. (1989), "A probabilistic heuristic for a computationally difficult set covering problem". Operations Research Letters, 8, 67–71.

Guinet, A. (1991), "Textile production systems: a sucession of non-identical parallel processor shops". Journal of the Operational Research Society, 42(8): 655-671.

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.

Laguna, M. e González-Velarde, J.L. (1991), "A search heuristic for just-in-time scheduling in parallel machines". Journal of Intelligent Manufacturing., 2, 253–260.

Lourenço, H.R.; Martin, O. e Stutzle, T. (2002), "Iterated Local Search". In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, pages 321–353. Kluwer Academic Publishers, Norwell, MA.

Marsh, J.D. e Montgomery, D.C. (1973), "Optimal procedures for scheduling Jobs with sequence-dependent changeover times on parallel processors". AIIE Technical Papers, 279-286.

Osman, I. e Potts, C. (1989), "Simulated annealing for permutation flow-shop scheduling". Omega, 17(6), 551–557.

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.

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.

Ruiz, R. e Stutzle, T. (2008), "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives". European Journal of Operational Research, 187(3), 1143–1159.

Tang, L.X. e Luo, J.X. (2006), "A new ILS algorithm for parallel machine scheduling problems". Journal of Intelligent Manufacturing., 17(5), 609–619.

Webster, S.T. (1997), "The complexity of scheduling job families about a common date". Operations Research Letters, 20(2), 65–74.