SPOLM 2009

ISSN 2175-6295 Rio de Janeiro- Brasil, 05 e 06 de agosto de 2009.

077/2009 - ITERATED LOCAL SEARCH COM PATH RELINKING APLICADO AO PROBLEMA DE PROGRAMAÇÃO DE TAREFAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA E DE RECURSOS

Edmar Hell Kampke
Universidade Federal de Viçosa - UFV
Avenida P.H. Rolfs s/n – Campus UFV – Viçosa, MG, 36570-000
edmar.kampke@ufv.br

José Elias Cláudio Arroyo
Universidade Federal de Viçosa - UFV
Avenida P.H. Rolfs s/n – Campus UFV – Viçosa, MG, 36570-000
jarroyo@dpi.ufv.br

Antônio Almeida de Barros Júnior
Universidade Federal de Viçosa - UFV
Avenida P.H. Rolfs s/n – Campus UFV – Viçosa, MG, 36570-000
antonio.almeida@ufv.br

RESUMO

Este trabalho aborda o problema de seqüenciamento de tarefas em máquinas paralelas, com tempos de preparação das máquinas dependentes da seqüência e do número de recursos utilizados. A característica deste problema é que o tempo de preparação não é determinado apenas pela máquina e pela seqüência das tarefas, mas também pela quantidade de recursos associados que varia entre um valor mínimo e máximo. Dada a complexidade combinatória do problema, propõe-se um algoritmo baseado na metaheurística *Iterated Local Search* (ILS). No algoritmo é utilizado uma estratégia de intensificação baseada na técnica *Path Relinking*, que consiste em explorar trajetórias que conectam soluções de alta qualidade encontradas pelo ILS. Os resultados obtidos pelo algoritmo proposto são comparados com os melhores resultados disponibilizados na literatura e apresentam bom desempenho em termos de qualidade das soluções.

Palavras-Chaves: Metaheurísticas; Programação de Tarefas; Tempo de Preparação; Iterated Local Search; Path Relinking.

ABSTRACT

This paper addresses an unrelated parallel machine problem with machine and job sequence dependent setup times. The characteristic of this problem is that the amount of setup time do not only depend on the machine and job sequence, but also on a number of resources assigned, which can vary between a minimum and a maximum. Due to the combinatorial complexity of this problem, we propose an algorithm based on metaheuristic Iterated Local Search (ILS). The algorithm uses an intensification strategy based on the Path Relinking technique which consists in exploring paths between elite solutions found by the ILS. The results obtained by the proposed algorithm are compared with the best results available in the literature and show good performance in terms of quality of solutions.

Keywords: Metaheuristcs; Scheduling; Setup Time; Iterated Local Search; Path Relinking.

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, seqüenciar 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).

O problema de PTMP considerado neste trabalho foi abordado pela primeira vez por Ruiz e Andrés (2007). Este problema 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}. 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 problema PTMPSR 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 problema PTMPSR 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 e com o número de recursos utilizados, que neste caso, pode ser 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 $C_{14} = 11$ e $C_{12} = 39$, respectivamente. O *setup time* entre estas tarefas é $S_{142} = 15$, 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 $C_{23} = 7$ e $C_{21} = 25$, respectivamente. O *setup time* entre estas tarefas é $S_{231} = 7$ com utilização de $R_{231} = 4$ unidades de recursos.

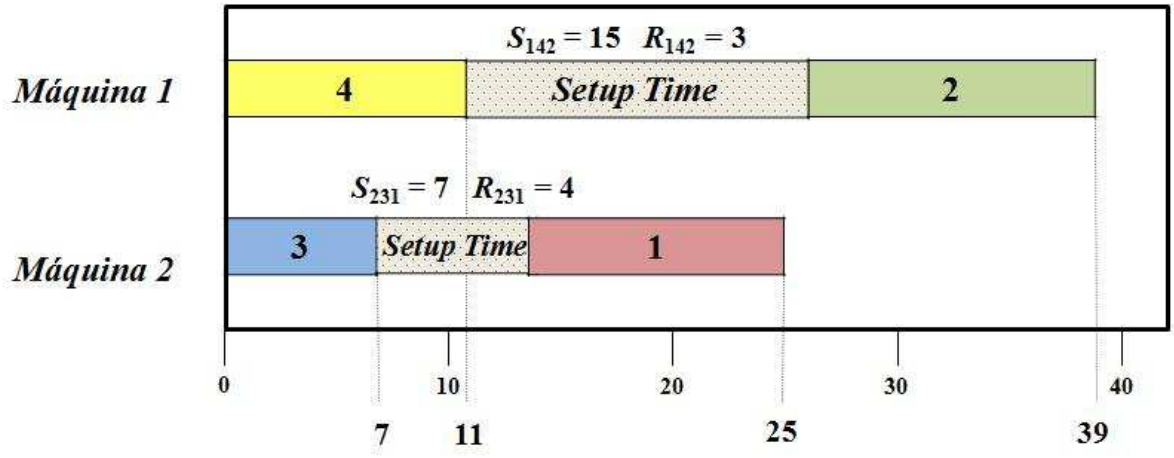

Figura 1: Exemplo de Solução

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 problema PTMPSR. A diferença entre as heurísticas está no critério de ordenação das tarefas,

utilizado 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 (2007) verificaram que a heurística DJASA apresenta os melhores resultados.

Neste artigo é proposto um algoritmo baseado na metaheurística ILS (*Iterated Local Search*) para resolver de forma aproximada o problema PTMPSR. O algoritmo utiliza também a técnica Path Relinking como estratégia de intensificação, que consiste em explorar trajetórias que conectam soluções de alta qualidade. Para testar o algoritmo, utiliza-se 720 problemas disponíveis na literatura. Os resultados obtidos são comparados com os melhores resultados encontrados pelas heurísticas de Ruiz e Andrés (2007).

2. ILS COM PATH RELINKING PARA O PROBLEMA PTMPSR

Iterated Local Search (ILS) é um algoritmo heurístico, 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).

Na Figura 2, apresenta-se um pseudocódigo genérico do algoritmo 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. O algoritmo ILS é fortemente dependente da solução inicial ótima local. Neste trabalho, para a geração da solução inicial, é utilizada uma heurística gulosa construtiva seguida de um algoritmo de busca local.

Os parâmetros a serem definidos no algoritmo ILS são o critério de parada, que geralmente é o número de iterações do algoritmo, a quantidade de elementos perturbados e o fator que define a intensidade do critério de aceitação.

```
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 2: Pseudocódigo genérico da heurística ILS

A seguir são descritos os procedimentos implementados pelo algoritmo ILS e pela estratégia de intensificação *Path Relinking*. Também são descritos o critério de parada e os parâmetros utilizados neste trabalho.

2.1. CONSTRUÇÃO DA SOLUÇÃO INICIAL

Nesta etapa uma solução (sequência de tarefas) para o problema PTMPSR é gerada. Utiliza-se a heurística gulosa construtiva DJASA, proposta por Ruiz e Andrés (2007). A heurística inicia com uma sequência vazia, e iterativamente adiciona-se uma tarefa à sequência parcial. Para escolher a tarefa a ser adicionada, são feitas simulações de inclusão, de cada tarefa não sequenciada, em todas as máquinas da sequência parcial. As tarefas não sequenciadas, são então ordenadas numa lista, de forma crescente pelo incremento calculado durante as simulações, que elas proporcionarão ao valor da função objetivo *Z*. A primeira tarefa da lista ordenada, ou seja, a tarefa que proporciona o menor incremento no valor da função objetivo é adicionada à sequência parcial na respectiva máquina que garante este incremento mínimo. A heurística DJASA finaliza quando a sequência conter todas as *n* tarefas. A Figura 3 mostra o algoritmo de construção da solução inicial.

A sequência construida pela heurística DJASA é melhorada utilizando um procedimento de busca local detalhado na seção 2.3.

2.2. 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.

```
Procedimento Construção_Solução_Inicial
1   L ← {t1,...,tn};                          // Lista de todas as tarefas
2   Sequência ← { };                          // Sequencia vazia
3   enquanto ( | Sequência | < n ) faça
4       Para cada tarefa ti de L, calcula-se o valor de Z ao adicionar ti na Sequência
5       Ordena_Lista(L);                      //Ordena L pelo valor de Z calculado
6       Sequência ← Sequência ∪ {t1};         //Adiciona a 1ª tarefa de L na Sequência
7       L ← L - {t1};                         //Remove a tarefa t1 de L
8   fim-enquanto;
9   s ← Busca_Local(Sequência);
10  retorne s;
fim Construção_Solução_Inicial;
```

Figura 3: Algoritmo de Construção da Solução Inicial

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 $k = n+m-1$, é a sequência original s_1 sem as tarefas removidas.

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 4 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.

```
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 4: Pseudocódigo do procedimento de Perturbação

2.3. 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 seja possível melhorar a solução atual s_1, ou seja, quando s_1 é um ótimo local. Neste trabalho foi utilizado 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).

2.4. 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, conseqüentemente, uma rápida estagnação das soluções avaliadas. Neste trabalho utiliza-se um critério de aceitação similar ao usado pela heurística *Simulated Annealing*. No entanto, o valor do parâmetro *Temperatura* é constante:

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

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 Stützle (2008). Na Figura 5 apresenta-se o pseudocódigo do *Critério de Aceitação*. Neste pseudocódigo nota-se que 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 ($s_1 \leftarrow s_2$), conforme apresentado no passo 2. Se s_2 melhorar a melhor solução atual s, então a solução s é atualizada no passo 4.

No caso de s_2 ser pior que s_1, s_2 é aceita com uma probabilidade definida da seguinte maneira:

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

onde *exp* é a função exponencial.

```
Procedimento Criterio_Aceitação(s, s1, s2, T );
1        se f(s2) < f(s1) então
2            s1 ← s2;
3            se f(s1) < f(s) então
4                s ← s1;
5            fim-se;
6        senão
7            r ← Número Aleatório no Intervalo [0,1];
8            se r < exp{-(f(s2) - f(s1))/Temperatura}então
9                s1 ← s2;
10           fim-se;
11       fim-se;
fim Criterio_Aceitação;
```

Figura 5: Pseudocódigo do Critério de Aceitação

2.5. INTENSIFICAÇÃO COM PATH RELINKING

A Reconexão de Caminhos (*Path Relinking*) é uma estratégia de intensificação e de soluções proposta por Glover (1996). Dado um par de soluções, denominadas origem e guia, o objetivo desta estratégia é construir iterativamente um caminho de soluções entre a solução origem e a solução guia, por meio de movimentos realizados na solução origem.

Neste trabalho, o *path relinking* mantém um conjunto de soluções elite *E*, de forma similar a Aiex *et al.* (2005). A solução origem s_s é obtida pela busca local. O procedimento inicia com a escolha aleatória da solução guia $s_g \in E$, desde que $s_g \neq s_s$. Através de movimentos de troca de tarefas em s_s, o *path relinking* tenta encontrar soluções melhores que s_s e s_g, que ainda não foram avaliadas.

Em cada etapa do procedimento, analisam-se todas as possíveis trocas na ordem das tarefas, desde que a troca incorpore 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 encontrada durante a execução da estratégia de intensificação. O procedimento continua até que s_s e s_g se tornem iguais.

2.6. ALGORITMO ILS COM PATH RELINKING

No algoritmo proposto, o *path relinking* está acoplado ao algoritmo ILS, sendo aplicado como refinamento de ótimos locais. Durante a execução do ILS, as soluções de alta qualidade são armazenadas no conjunto *E*. O algoritmo tem como entrada o parâmetro E_{Tam} (tamanho máximo do conjunto *E*), o critério de parada e os parâmetros *d* e *T*, utilizados no procedimento de perturbação e no critério de aceitação, respectivamente.

```
Procedimento ILSPR (E_Tam, CritérioParada, d, T)
1    E = ∅;
2    s1 ← Construção_Solução_Inicial;
3    s ← s1;
4    enquanto não CritérioParada faça
5        s2 ← Perturbação(s1, d);
6        s2 ← Busca_Local(s2);
7        se |E| = E_Tam então
8            s3 ← Escolha aleatoriamente uma solução elite de E;
```

```
9          s4 ← Path_Relinking(s2, s3);
10         s5 ← Path_Relinking(s3, s2);
11         s6 ← Melhor_Solucao(s4, s5);
12         se s6 ∉ E e s6 ≠ s2 então
13            s2 ← Busca_Local(s6);
14         senão
15            s2 ← s6;
16         fim-se
17         Atualiza_Elite(s2);
18      senão
19         E ← E ∪ {s2};
20      fim-se
21      Criterio_Aceitação(s, s1, s2, T);
22   fim-enquanto;
23   s ← Melhor_Solucao(E);
24   retorne s;
fim ILSPR;
```

Figura 6: Estrutura Genérica do Algoritmo ILSPR.

No conjunto E não se permite a inclusão de soluções repetidas. Durante a execução do ILS, enquanto $|E|$ for menor que E_{Tam}, a solução obtida pela fase de busca local é inserida em E, desde que seja diferente das soluções já contidas neste conjunto. Quando $|E| = E_{Tam}$, então aplica-se a estratégia de intensificação com *path relinking*, construindo dois caminhos, um de s_s para s_g, e outro de s_g para s_s. A melhor solução obtida na construção desses caminhos é comparada com a pior solução contida no conjunto E, e se for melhor, a pior solução de E é substituída por esta. Ou seja, o conjunto E sempre terá as E_{Tam} melhores soluções de elite.

Na Figura 6 apresenta-se a estrutura genérica do algoritmo proposto, ILSPR, que combina características dos procedimentos ILS e *path relinking*. O passo 1 demonstra que o conjunto de soluções elite é inicializado vazio. No passo 7 verifica-se se o conjunto E já foi totalmente preenchido. No passo 8 escolhe-se aleatoriamente a solução s_3 para ser a solução guia. Em seguida, nos passos 9 e 10, é aplicado a estratégia de intensificação, através da construção de caminhos entre a solução origem e a solução guia. No passo 11 a melhor das soluções s_4 e s_5, retornadas pelo *path relinking*, é armazenada em s_6. Caso s_6 não faça parte do conjunto elite e seja diferente de s_2 (obtida pela busca local), aplica-se a busca local em s_6 e armazena-se em s_2 a solução retornada (passo 13). No passo 17, se for o caso, o conjunto E é atualizado através da substituição da pior solução de E por s_2. A melhor solução do conjunto elite é retornada como solução do método ILSPR (passos 23 e 24).

2.7. CRITÉRIO DE PARADA

Na implementação dos algoritmos ILS e ILSPR, 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.

2.8. PARÂMETROS DOS ALGORITMOS

No algoritmo ILSPR o tamanho do conjunto de soluções elite foi definido por E_{Tam}=10 e em ambos os algoritmos, ILS e ILSPR, 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 $T = 0{,}5$.

3. RESULTADOS COMPUTACIONAIS

Neste artigo, testa-se o desempenho dos algoritmos ILS e ILS com Path Relinking (ILSPR) na resolução do problema PTMPSR. 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 ambos os grupos de problemas, os tempos de processamento das tarefas são gerados aleatoriamente (com distribuição uniforme) no intervalo [1,99]. Os recursos (mínimo R^-; máximo R^+) e os *setup times* (mínimo S^-; máximos S^+) são também gerados aleatoriamente (com distribuição uniforme) dentro de intervalos pré-definidos.

Para todos os problemas, os parâmetros λ e δ, que representam os pesos dos critérios otimizados, foram os mesmos utilizados por Ruiz e Andrés (2007): λ=50 e δ=1.

Os algoritmos foram implementados na linguagem de programação Java com a versão 1.6 e executados 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.

Os resultados obtidos pelos algoritmos são comparados com os melhores resultados disponibilizados por 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) proposto por 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. Para todos os problemas com n=10 somente foram obtidas soluções aproximadas, devido ao limite do tempo de execução do CPLEX.

Tabela 1: Comparação de resultados dos algoritmos ILS e ILSPR com os resultados do MMPI (CPLEX 9.1)

Problemas				***Médias de Z***			***Melhora (%)***	
n	***m***	$S^-;S^+$	$R^-;R^+$	***CPLEX***	***ILS***	***ILSPR***	***ILS***	***ILSPR***
6	3	[1,50];[50,100]	[1,3];[3;5]	6.318	6.318	6.318	**0,00**	**0,00**
			[1,5];[5;10]	7.169	7.169	7.169	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	7.986	7.986	7.986	**0,00**	**0,00**
			[1,5];[5;10]	8.057	8.057	8.057	**0,00**	**0,00**
	4	[1,50];[50,100]	[1,3];[3;5]	4.360	4.365	4.360	**-0,11**	**0,00**
			[1,5];[5;10]	4.200	4.200	4.200	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	5.168	5.168	5.168	**0,00**	**0,00**
			[1,5];[5;10]	5.238	5.238	5.238	**0,00**	**0,00**
	5	[1,50];[50,100]	[1,3];[3;5]	2.738	2.738	2.738	**0,00**	**0,00**
			[1,5];[5;10]	2.473	2.473	2.473	**0,00**	**0,00**
		[50,100];[100,150]	[1,3];[3;5]	3.148	3.148	3.148	**0,00**	**0,00**
			[1,5];[5;10]	3.128	3.128	3.128	**0,00**	**0,00**
8	3	[1,50];[50,100]	[1,3];[3;5]	11.029	11.029	11.029	**0,00**	**0,00**
			[1,5];[5;10]	12.236	12.288	12.262	**-0,42**	**-0,21**
		[50,100];[100,150]	[1,3];[3;5]	14.811	14.811	14.811	**0,00**	**0,00**
			[1,5];[5;10]	16.047	16.047	16.047	**0,00**	**0,00**
	4	[1,50];[50,100]	[1,3];[3;5]	8.287	8.287	8.287	**0,00**	**0,00**
			[1,5];[5;10]	8.772	8.933	8.896	**-1,84**	**-1,41**
		[50,100];[100,150]	[1,3];[3;5]	9.965	9.965	9.965	**0,00**	**0,00**
			[1,5];[5;10]	10.709	10.781	10.729	**-0,67**	**-0,19**
	5	[1,50];[50,100]	[1,3];[3;5]	5.748	5.748	5.748	**0,00**	**0,00**
			[1,5];[5;10]	6.459	6.498	6.462	**-0,60**	**-0,05**
		[50,100];[100,150]	[1,3];[3;5]	7.193	7.193	7.193	**0,00**	**0,00**
			[1,5];[5;10]	7.916	7.916	7.916	**0,00**	**0,00**
10	3	[1,50];[50,100]	[1,3];[3;5]	16.302	16.079	16.079	**1,37**	**1,37**

			[1,5];[5;10]	18.590	18.303	18.303	**1,54**	**1,54**
		[50,100];[100,150]	[1,3];[3;5]	23.441	22.866	22.866	**2,45**	**2,45**
			[1,5];[5;10]	26.326	25.958	25.912	1,40	**1,57**
	4	[1,50];[50,100]	[1,3];[3;5]	12.993	12.807	12.807	**1,43**	**1,43**
			[1,5];[5;10]	13.414	13.292	13.292	**0,91**	**0,91**
		[50,100];[100,150]	[1,3];[3;5]	16.462	16.099	16.092	2,21	**2,25**
			[1,5];[5;10]	17.245	17.159	17.056	0,50	**1,10**
	5	[1,50];[50,100]	[1,3];[3;5]	9.409	9.139	9.139	**2,87**	**2,87**
			[1,5];[5;10]	10.165	9.984	9.966	1,78	**1,96**
		[50,100];[100,150]	[1,3];[3;5]	12.314	11.848	11.848	**3,78**	**3,78**
			[1,5];[5;10]	12.785	12.194	12.177	4,62	**4,76**
Média				-	-	-	**0,59**	**0,67**

A Tabela 1 exibe os resultados encontrados pelo software CPLEX e pelos algoritmos: ILS e ILSPR, para os 360 problemas do grupo *small*. Nesta Tabela apresentam-se as médias dos valores da função objetivo *Z* a cada 10 problemas que possuem as mesmas dimensões. Nas colunas intituladas *Melhora* são apresentados os percentuais de melhora dos algoritmos, *ILS* e *ILSPR*, sobre o software CPLEX. Estes percentuais são resultados da aplicação das seguintes fórmulas:

$$\text{Melhora ILS (\%)} = \frac{100 \times (\text{CPLEX} - \text{ILS})}{\text{CPLEX}}; \qquad \text{Melhora ILSPR (\%)} = \frac{100 \times (\text{CPLEX} - \text{ILSPR})}{\text{CPLEX}}$$

Para os problemas do grupo *small*, os resultados obtidos pelos algoritmos ILS e ILSPR foram bastante satisfatórios. Pela Tabela 1, observa-se que entre 12 grupos de problemas com $n = 6$ tarefas, em apenas um o algoritmo ILS não encontrou a solução ótima, enquanto o algoritmo ILSPR encontrou a solução ótima em todos os 120 problemas. Nos 120 problemas com $n = 8$ tarefas, as heurísticas ILS e ILSPR encontraram a solução ótima em 110 e 112 problemas, respectivamente. Para os 120 problemas restantes, com $n = 10$ tarefas, ambas as heurísticas, ILS e ILSPR, encontraram soluções superiores em 82 e 83 problemas, respectivamente, obtendo um percentual médio de melhoria de 2,07% (ILS) e 2,17% (ILSPR).

Tabela 2: Comparação de resultados dos algoritmos ILS e ILSPR com os melhores resultados da literatura

Problemas				***Médias de Z***			***Melhora (%)***	
n	***m***	$S^-;S^+$	$R^-;R^+$	***Melhores Resultados Heurísticos***	***ILS***	***ILSPR***	***ILS***	***ILSPR***
50	10	[1,50];[50,100]	[1,3];[3;5]	92.615	81.961	81.914	11,50	**11,55**
			[1,5];[5;10]	112.832	101.101	100.895	10,40	**10,58**
		[50,100];[100,150]	[1,3];[3;5]	143.256	132.090	131.748	7,79	**8,03**
			[1,5];[5;10]	164.401	151.802	151.956	**7,66**	7,57
	15	[1,50];[50,100]	[1,3];[3;5]	67.979	59.081	59.219	**13,09**	12,89
			[1,5];[5;10]	74.990	67.973	68.023	**9,36**	9,29
		[50,100];[100,150]	[1,3];[3;5]	97.742	90.421	90.751	**7,49**	7,15
			[1,5];[5;10]	105.534	98.558	98.706	**6,61**	6,47
	20	[1,50];[50,100]	[1,3];[3;5]	49.928	45.295	45.152	9,28	**9,57**
			[1,5];[5;10]	53.859	48.110	48.134	**10,67**	10,63
		[50,100];[100,150]	[1,3];[3;5]	70.002	65.412	65.515	**6,56**	6,41
			[1,5];[5;10]	74.022	68.728	68.734	**7,15**	7,14
75	10	[1,50];[50,100]	[1,3];[3;5]	166.415	147.055	147.450	**11,63**	11,40
			[1,5];[5;10]	236.000	195.235	195.003	17,27	**17,37**
		[50,100];[100,150]	[1,3];[3;5]	286.680	268.341	268.109	6,40	**6,48**
			[1,5];[5;10]	354.873	318.812	320.549	**10,16**	9,67
	15	[1,50];[50,100]	[1,3];[3;5]	130.347	115.036	115.394	**11,75**	11,47
			[1,5];[5;10]	159.490	143.760	143.755	9,86	**9,87**
		[50,100];[100,150]	[1,3];[3;5]	205.400	191.125	190.454	6,95	**7,28**
			[1,5];[5;10]	236.376	223.131	222.499	5,60	**5,87**
	20	[1,50];[50,100]	[1,3];[3;5]	107.804	92.958	93.279	**13,77**	13,47
			[1,5];[5;10]	120.296	109.588	109.383	8,90	**9,07**

		[50,100];[100,150]	[1,3];[3;5]	160.067	147.949	147.667	7,57	**7,75**
			[1,5];[5;10]	172.102	163.691	163.888	**4,89**	4,77
100	10	[1,50];[50,100]	[1,3];[3;5]	246.576	220.202	220.301	**10,70**	10,66
			[1,5];[5;10]	359.014	305.310	304.140	14,96	**15,28**
		[50,100];[100,150]	[1,3];[3;5]	469.931	445.769	445.974	**5,14**	5,10
			[1,5];[5;10]	585.176	528.246	527.512	9,73	**9,85**
	15	[1,50];[50,100]	[1,3];[3;5]	195.793	176.067	176.308	**10,07**	9,95
			[1,5];[5;10]	270.150	234.273	233.794	13,28	**13,46**
		[50,100];[100,150]	[1,3];[3;5]	336.943	318.463	318.618	**5,48**	5,44
			[1,5];[5;10]	409.763	379.459	380.591	**7,40**	7,12
	20	[1,50];[50,100]	[1,3];[3;5]	167.631	149.495	148.795	10,82	**11,24**
			[1,5];[5;10]	206.661	188.242	187.704	8,91	**9,17**
		[50,100];[100,150]	[1,3];[3;5]	267.809	251.372	250.608	6,14	**6,42**
			[1,5];[5;10]	306.226	293.397	292.978	4,19	**4,33**
Média				-	-	-	**9,14**	**9,16**

Na Tabela 2 comparam-se os resultados obtidos pelos algoritmos ILS e ILSPR, para os 360 problemas do grupo *large*, com os melhores resultados das heurísticas construtivas propostas por Ruiz e Andrés (2007). Da mesma forma que na Tabela 1, apresentam-se as médias dos valores da função objetivo *Z*, para cada grupo de 10 problemas que possuem as mesmas dimensões, e as colunas intituladas *Melhora* apresentam os percentuais de melhora dos algoritmos, *ILS* e *ILSPR*, sobre os melhores resultados heurísticos da literatura.

Para os 360 problemas do grupo *large*, os algoritmos ILS e ILSPR obtiveram, em todos os casos, resultados superiores em relação aos melhores resultados heurísticos disponibilizados na literatura. Observa-se pela Tabela 2, que em 18 sub-grupos de problemas o algoritmo ILSPR superou os resultados obtidos pelo algoritmo ILS. Comparando os resultados obtidos pelos algoritmos ILS e ILSPR, para cada um dos 360 problemas do grupo *large*, foi observado que em 188 problemas o resultado do algoritmo ILSPR superou o algoritmo ILS. Em 3 problemas os resultados obtidos por ambas os algoritmos foram idênticos e o algoritmo ILS mostrou resultado superior, com relação o algoritmo ILSPR, nos 169 problemas restantes. Na tabela 2, observa-se também que os menores valores percentuais de melhoria para os algoritmos ILS e ILSPR foram: 4,19% e 4,33%, respectivamente. Enquanto os maiores valores foram: 14,96% e 15,28%.

Na Figura 7 ilustra-se a média de melhoria do valor da função objetivo a medida que o tempo computacional do algoritmo ILS é incrementado. Essa análise foi realizada para um conjunto de 30 problemas do conjunto *large* escolhidos aleatoriamente. No gráfico, nota-se que as soluções obtidas apresentam um grande melhoramento na primeira metade do tempo de execução. Na segunda metade nota-se um melhoramento menor. Dentre os 30 problemas analisados, apenas 7 conseguiram melhorar a solução na segunda metade do tempo de execução. Esta análise foi importante para determinar a condição de parada do algoritmo.

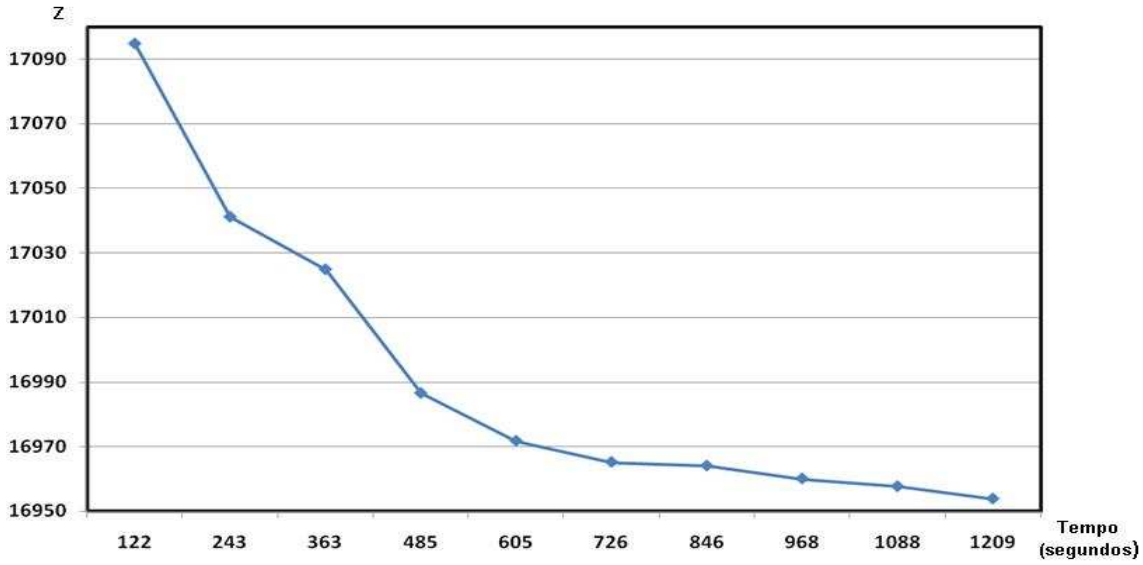

Figura 7: Melhoramento das soluções na execução do algoritmo ILS.

4. CONCLUSÕES

Neste trabalho foi proposto um algoritmo *Iterated Local Search* (ILS) na qual é testado o comportamento da técnica *Path Relinking* (ILSPR) 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 do algoritmo ILS. O desempenho do algoritmo proposto foi testado em 720 problemas, incluindo de pequeno e grande porte. As soluções obtidas pelos algoritmos foram comparadas com as melhores soluções encontradas na literatura. Para os problemas de grande porte, o algoritmo ILSPR obteve uma melhoria média de 9,16% com relação às melhores soluções heurísticas disponibilizadas na literatura, o que demonstra a eficácia do algoritmo. O algoritmo ILSPR, em média, obteve desempenho superior ao algoritmo ILS 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 o algoritmo ILS com *Simulated Annealing* ou *Greedy Randomized Adaptive Search Procedure* (GRASP).

5. 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.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] **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.

[2] **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.

[3] **Glover, F.** (1996), Tabu search and adaptive memory programming - advances, applications and challenges. In: Barr, R.S., Helgason, R.V. and Kenngington, J.L. editors. *Interfaces in Computer Science and Operations Research*, 1–75.

[4] **Guinet, A.** (1991), Textile production systems: A succession of non-identical parallel processor shops. *Journal of the Operational Research Society*,42, 655–671.

[5] **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.

[6] **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.

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

[8] **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.

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

[10] **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.

[11] **Ruiz, R. e Stützle, 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.

[12] **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.

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