ISSN 2175-6295 Rio de Janeiro- Brasil, 12 e 13 de agosto de 2010

UMA ABORDAGEM HEURÍSTICA VIA MULTIPROCESSAMENTO PARALELO PARA UMA CLASSE DE PROBLEMAS DE SEQUENCIAMENTO EM UMA MÁQUINA

Frederico Augusto de Cezar Almeida Gonçalves[1], Marcone Jamilson Freitas Souza[2], Sérgio Ricardo de Souza[1], Leandro Augusto de Araújo Silva[2]

[1]Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG
Av. Amazonas, 7675 – Belo Horizonte, MG, CEP 30.510-000
zepfred@gmail.com, sergio@dppg.cefetmg.br

[2]Universidade Federal de Ouro Preto
Campus Universitário Morro do Cruzeiro – Ouro Preto, MG, CEP 35.400-000
marcone@iceb.ufop.br, leandrogattuso88@gmail.com

Resumo: Este trabalho tem seu foco no problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. São considerados tempos de preparação da máquina dependentes da sequência de produção, bem como a existência de janelas de entrega distintas. Para sua resolução, propõe-se um algoritmo heurístico de processamento paralelo. Esse algoritmo, denominado GTSPR-M, combina os procedimentos heurísticos GRASP, Variable Neighborhood Descent, Busca Tabu e Reconexão por Caminhos. O algoritmo proposto foi testado em um conjunto de problemas-teste da literatura e comparado com sua versão sequencial, bem como com um algoritmo genético. Os resultados computacionais mostraram que o GTSPR-M foi superior a esses algoritmos, tanto com relação à qualidade da solução final quanto em relação ao desvio médio e tempo de processamento na maioria dos casos.

Palavras-chave: Sequenciamento em uma máquina, Processamento Paralelo, Metaheurísticas.

Abstract: This work has its focus in the problem of scheduling single machine with earliness and tardiness penalties, distinct due windows and sequence dependent setup. A parallel processing algorithm to solve it is proposed. This algorithm, named GTSPR-M, combines the heuristic procedures GRASP, Variable Neighborhood Descent, Tabu Search and Path Relinking. The GTSPR-M was compared to its sequential version as well as a genetic algorithm, using a set of instances of the literature. Computational results showed that GTSPR-M outperforms these algorithms, with respect to the quality of the final solution, the average gap and processing time in most cases.

Keywords: Single machine scheduling, Parallel processing, Metaheuristics.

1 Introdução

Trata-se, neste trabalho, o problema de sequenciamento em uma máquina com minimização das penalidades por antecipação e atraso da produção (*Single Machine Scheduling for Minimizing Earliness and Tardiness Penalties*), em que há um conjunto de janelas de entrega distintas e tempos de preparação da máquina dependentes da sequência da produção.

Este problema, denotado por PSUMAA-JP, representa uma generalização dos problemas de sequenciamento em uma máquina com penalidades por antecipação e atraso. De fato, quando a janela de entrega de uma tarefa reduz-se a uma data de entrega, tem-se o problema com datas de entrega distintas. Caso as janelas sejam as mesmas para todas as tarefas e estas se reduzam a uma mesma data, tem-se o problema com datas comuns de entrega. Considerando-se tempo de preparação nulo, tem-se o problema de sequenciamento com tempo de preparação independente da sequência.

Em vista de sua dificuldade de solução, esse problema tem sido tratado na literatura por meio de algoritmos heurísticos, todos sequenciais. Assim são os algoritmos de Gomes Jr. *et al*. (2007), Souza *et al*. (2008), Rosa *et al*. (2009) e Ribeiro *et al*. (2009).

Com a presença de recursos *multicore* na maioria dos computadores atuais é possível explorar paralelismo de uma maneira muito mais simples do que com a utilização de um *cluster* envolvendo várias máquinas, possivelmente heterogêneas. A paralelização de algoritmos traz, entre outras vantagens, a possibilidade de melhorar a qualidade das soluções finais ou a redução do tempo computacional.

Assim, de forma a explorar os recursos de multiprocessamento dos computadores atuais, propõe-se um algoritmo heurístico paralelo para resolver o problema em questão. Em linhas gerais, ele funciona como segue. Na primeira fase, aplica-se GRASP para gerar uma solução inicial. Nessa fase, a solução é construída distribuindo-se o processamento entre vários *threads*, sendo estes processados em paralelo. A melhor solução encontrada por algum destes *threads* é utilizada como solução inicial para o algoritmo. A seguir, na segunda fase, esta solução é refinada pelo procedimento Busca Tabu (*Tabu Search* – TS). Neste procedimento, a geração de vizinhos bem como o refinamento destes são executados de forma paralela. Durante a execução do procedimento Busca Tabu também é construído um conjunto de boas soluções, diferenciadas entre si, o chamado Conjunto Elite (CE). Esse conjunto é utilizado na terceira fase, na qual é executado o procedimento Reconexão por Caminhos (*Path Relinking* – PR) como estratégia de pós-otimização. A Reconexão é aplicada a cada par de soluções do Conjunto Elite, sendo cada par executado em modo paralelo em um *thread*.

O restante deste trabalho está estruturado como segue. Na Seção 2 é feita uma breve revisão de trabalhos relacionados. Na Seção 3 o problema em questão é caracterizado. A Seção 4 apresenta o algoritmo heurístico proposto para resolver o PSUMAA-JP. Resultados computacionais são apresentados no capítulo 5. Finalmente, na Seção 6, conclui-se este trabalho e apresenta-se uma proposta para trabalhos futuros.

2 Trabalhos relacionados

Gomes Jr. *et al.* desenvolveram uma formulação de programação linear inteira mista para o PSUMAA-JP (2007). O modelo desenvolvido foi utilizado para resolver na otimalidade problemas de até 12 tarefas. Nesse trabalho também foi proposto um algoritmo heurístico, chamado de GILS-VND-RT, que combina os procedimentos heurísticos GRASP, *Iterated Local Search* (ILS) e *Variable Neighborhood Descent* (VND). Para cada sequência gerada pela heurística, foi desenvolvido um procedimento para determinar a data ótima de conclusão do processamento de cada tarefa. Tal procedimento foi adaptado de Wan e Yen (2002), incluindo-se no tempo de processamento de uma tarefa o tempo de preparação da máquina, já que quando ele

é acionado, já se conhece a sequência de produção.

Souza *et al.* (2008) desenvolveram um algoritmo híbrido sequencial de três fases, nomeado GTSPR, baseado em GRASP, VND, Busca Tabu e Reconexão por caminhos para o PSUMAA-JP. Na primeira fase os autores utilizam um procedimento baseado em GRASP para gerar a solução inicial. As soluções geradas neste método são refinadas por um procedimento baseado em VND. Na segunda fase, a melhor solução construída pelo procedimento GRASP é, então, refinada por um procedimento baseado em Busca Tabu. Durante a execução da Busca Tabu é formado um conjunto elite com soluções de boa qualidade e diferenciadas entre si. Na terceira e última fase, como estratégia de pós-otimização, é aplicado o procedimento Reconexão por Caminhos a todos os pares de solução do conjunto elite. Esse algoritmo se mostrou superior ao de Gomes Jr. *et. al.* (2007) quando comparado em termos da qualidade da solução final e tempo de processamento.

Ribeiro *et al.* (2009) propuseram um Algoritmo Genético Adaptativo, denominado AGA, para a resolução do PSUMAA-JP. A população inicial é formada com base na aplicação da fase de construção GRASP, tendo como função guia várias regras de despacho. Para cada indivíduo gerado é utilizado o procedimento PDDOIP de Gomes Jr. *et al.* (2007) para determinar a data ótima de início de processamento de cada tarefa na sequência dada. São utilizados cinco operadores de cruzamento e um de mutação. Inicialmente, cada um dos operadores tem a mesma chance de ser escolhido. No entanto, de cinco em cinco gerações, é atualizada essa probabilidade, sendo que aos operadores de cruzamento com melhor desempenho são atribuídas as maiores probabilidades. Essa atualização é feita seguindo o procedimento GRASP Reativo (RESENDE e RIBEIRO, 2010). Durante as cinco gerações é formado um grupo elite com as melhores soluções geradas por cada operador. Após a atualização das probabilidades de aplicação de cada operador, esses integrantes do grupo elite são submetidos a uma busca local e, em seguida, à Reconexão por Caminhos, com o objetivo de encontrar soluções intermediárias de melhor qualidade. O AGA foi capaz de produzir soluções de melhor qualidade que o algoritmo de Gomes Jr. *et al*. (2007).

Rosa *et al.* (2009) desenvolveram dois modelos de programação linear inteira mista para o PSUMAA-JP. Dentre as duas novas formulações propostas nesse trabalho, uma delas se trata de um aperfeiçoamento do modelo de Gomes Jr. *et al*. (2007), e a outra, uma formulação indexada no tempo. Segundo os autores, com a formulação indexada no tempo foi possível obter um maior número de soluções ótimas, assim como um menor tempo de processamento para alcançar a otimalidade. Em Rosa (2009) foi proposto, também, um algoritmo heurístico de duas fases baseado em GRASP, VND e no princípio da otimalidade próxima. Esse algoritmo heurístico, nomeado GPV, obteve tempos computacionais inferiores aos da literatura e se mostrou competitivo com os algoritmos existentes até então.

Com relação à utilização de algoritmos paralelos, de nosso conhecimento, nenhuma aplicação ainda foi desenvolvida para tratar o problema em questão.

3 Caracterização do problema

O PSUMAA-JP possui as seguintes características: (a) Uma máquina deve processar um conjunto de n tarefas; (b) Cada tarefa i possui um tempo de processamento P_i . O término de cada tarefa i é delimitado pela janela de entrega $[E_i, T_i]$, onde E_i e T_i indicam, respectivamente, a data inicial e a data final desejadas para o término de seu processamento. Se a tarefa i for finalizada antes de E_i, há um custo α_i por unidade de tempo de antecipação. Caso a tarefa seja finalizada após T_i, então há um custo β_i por unidade de tempo de atraso. Considera-se que não há custo caso as tarefas sejam concluídas dentro da janela de entrega; (d) A máquina pode executar no máximo uma tarefa por vez. Seu funcionamento é não preemptível, ou seja, uma vez iniciado o processamento de uma tarefa, não é permitida a sua interrupção; (e) Todas as tarefas estão

disponíveis para processamento na data 0; (f) Entre duas tarefas *i* e *j* imediatamente consecutivas é necessário um tempo S_{ij} de preparação da máquina, chamado tempo de *setup*. Assume-se que a máquina não necessita de preparação para processamento da primeira tarefa, isto é, a primeira tarefa que será processada na máquina tem tempo de preparação igual a 0; (g) É permitido tempo ocioso entre o processamento de duas tarefas imediatamente consecutivas.

O objetivo é determinar a sequência que minimiza a penalidade total por antecipação e atraso de produção.

4 Metodologia

4.1 Representação da solução, Vizinhança e Função de avaliação

Representa-se uma solução do PSUMAA-JP por um vetor *v* de *n* tarefas, em que cada posição *i* de *v* indica a ordem em que a tarefa v_i será executada.

Explora-se o espaço de soluções por meio de três tipos de movimentos: troca da ordem de processamento de duas tarefas, realocação de uma tarefa para outra posição e realocação de um bloco de *k* tarefas, com $2 \leq k \leq n\text{-}2$. Esses movimentos definem, respectivamente, as estruturas de vizinhança N^T, N^R e N^{Or}.

Uma solução *v* é avaliada pela função *f* a seguir, a qual deve ser minimizada:

$$f(v) = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \quad (1)$$

Na Eq. (1), $e_i \geq 0$ e $t_i \geq 0$ representam, respectivamente, o tempo de antecipação e atraso da tarefa *i* e α_i e β_i são as penalidades respectivas. O procedimento PDDOIP (Gomes Jr. *et al.,* 2007) é utilizado para determinar as datas ótimas para início do processamento das tarefas das sequências e, assim, calcular a quantidade de antecipação e atraso relativa a cada tarefa.

4.2 Algoritmo GTSPR-M

Para resolver o PSUMAA-JP, propõe-se um algoritmo heurístico de três fases, nomeado GTSPR-M, cujo pseudocódigo é apresentado no Algoritmo 1.

Algoritmo 1: GTSPR-M

Entrada: $N_{processors}$, Γ, *PrazoTABU*, *difElite*, *GRASPmax*, *MRDmax*, *TABUmax*
Saída: v
1 **início**
2 $T \leftarrow \emptyset$
3 $CE \leftarrow \emptyset$
4 $v_0 \leftarrow$ GRASP-M($N_{processors}$, Γ, *GRASPmax*, *MRDmax*)
5 $v_1 \leftarrow$ TABU-M($N_{processors}$, v_0, *PrazoTABU*, T, *difElite*, *CE*, *TABUmax*, *MRDmax*)
6 $v \leftarrow$ PR-M($N_{processors}$, v_1, *CE*)
7 **Retorne** v
8 **fim**

Na primeira fase (linha 4), constrói-se uma solução inicial. A seguir (linha 5), a solução resultante desta fase é refinada por um algoritmo baseado em Busca Tabu (vide seção 4.4). Como pós-otimização (linha 6) é aplicada a Reconexão por Caminhos (vide seção 4.5). Cada uma dessas fases é detalhada a seguir.

4.3 Geração da Solução Inicial

A solução inicial é gerada pelo procedimento GRASP-M, descrito no Algoritmo 2. Esse método usa a heurística *Greedy Randomized Adaptive Search Procedures* – GRASP

(Resende e Ribeiro, 2010), tendo como método de busca local a Descida em Vizinhança Variável (*Variable Neighborhood Search* – VND) (Mladenović e Hansen, 1997). Além disso, o método também aplica uma estratégia de distribuição do processamento, que consiste em dividir as *GRASPmax* iterações do método entre os $N_{processors}$ núcleos de processamento disponíveis. Essa estratégia é adotada para que o mesmo possa ser executado em modo paralelo, e, assim, tirar proveitos de uma arquitetura com vários núcleos de processamento. De acordo com Resende e Ribeiro (2005), a estratégia adotada é categorizada como *multiple-walk independent-thread*.

O procedimento executado no Algoritmo 2, linha 5, é o que efetivamente constrói uma solução. Seu funcionamento é descrito no Algoritmo 3. Nele, Γ é um conjunto de parâmetros γ que definem o tamanho da Lista Restrita de Candidatos do GRASP, *NewGRASPmax* representa o número máximo de iterações GRASP executadas em um *thread*, MRDmax é o número máximo de iterações do método MRD. A variável *R* indica a aplicação da regra de despacho *Earliest Due Date* (EDD) sobre a data de início E_i da janela de entrega da tarefa *i* nas iterações ímpares ou sobre a data de término T_i dessa tarefa nas iterações pares. O procedimento de construção é o mesmo de Souza *et al*. (2008).

Algoritmo 3: NOVO-PROCESSO-GRASP

```
   Entrada: Γ, NewGRASPmax, MRDmax
   Saída: v_min
 1 início
 2     f_min ← ∞
 3     para i=1 até NewGRASPmax faça
 4         Escolha γ ∈ Γ
 5         Escolha R ∈ {EDD(E), EDD(T)}
 6         v ← Constroi_Solucao(R, γ)
 7         v ← Refinamento_1(v, MRDmax)
 8         se (f(v) < f_min) então
 9             v_min ← v;  f_min ← f(v)
10         fim
11     fim
12     v_min ← Refinamento_2(v_min, MRDmax)
13     Retorne v_min
14 fim
```

Na fase de refinamento do procedimento NOVO-PROCESSO-GRASP, são utilizados dois processos diferentes de refinamento. No primeiro, Refinamento$_1$, a execução do método varia entre movimentos de realocação e troca. Desta forma, explora-se o espaço de soluções segundo as vizinhanças N^R e N^T, nesta ordem. Inicialmente a solução corrente passa por uma descida randômica com movimentos de realocação. Para tanto, alguma tarefa J_i da sequência é sorteada aleatoriamente, assim como a posição *j* para onde ela será realocada. Se após a realocação, a solução gerada for melhor avaliada que a solução corrente segundo a função de avaliação, a solução é aceita e passa a ser a solução corrente. Decorridas *MRDmax* iterações sem melhora da solução corrente, o movimento de realocação é interrompido e sobre a solução corrente agora é aplicada uma descida randômica com movimentos de troca. Assim, duas tarefas J_i e J_j são sorteadas aleatoriamente e trocadas de posição. Se a troca produzir uma solução melhor segundo a função de avaliação, a solução é aceita como solução corrente e o refinamento com movimentos de troca é interrompido, retornando-se ao refinamento usando realocação do Algoritmo. Se não houver uma melhora da solução em *MRDmax* iterações, o método encerra a execução.

Ao final do procedimento NOVO-PROCESSO-GRASP, um segundo processo de refinamento é aplicado. Nesse processo, o procedimento Refinamento$_2$ explora o espaço de soluções por meio de movimentos de troca, realocação e movimentos *Or*. A solução é refinada segundo a seguinte estratégia: 1) descida randômica com movimentos de realocação; 2) descida randômica com movimentos de troca; 3) descida randômica com movimentos *Or* de realocação de um bloco de k tarefas ($2 \leq k \leq 3$); 4) descida completa com movimentos de realocação; 5) descida completa com movimentos de troca; 6) descida completa com movimentos *Or* de realocação de um bloco de k tarefas ($2 \leq k \leq n$-2). Nas descidas com realocação de k tarefas contíguas, inicialmente k assume seu valor mínimo; não havendo melhora, o valor de k é incrementado progressivamente até atingir seu limite máximo. Ao final da execução do procedimento, um ótimo local com relação às vizinhanças N^R, N^T e N^{Or} é retornado.

4.4 Busca Tabu

O procedimento Busca Tabu (Glover e Laguna, 1997) inicia sua execução a partir da solução retornada pelo procedimento GRASP-M. A cada iteração, o espaço de soluções é explorado alternando-se entre as vizinhanças N^R e N^T, nesta ordem. Após todos os vizinhos serem analisados, é aceito o melhor vizinho que não seja tabu, ou se tabu, que atenda a condição de aspiração. Para atender a esta condição, o vizinho considerado tabu deve ser melhor avaliado que a melhor solução existente até então segundo a função de avaliação, isto é, adota-se a condição de aspiração por objetivo global. O vizinho escolhido é, então, aceito como a solução corrente. Esta escolha não implica necessariamente em uma solução melhor que a solução corrente, uma vez que o melhor vizinho não tabu da solução corrente pode ser uma solução pior. Essa condição de piora permite que o método consiga escapar da armadilha de um ótimo local e explorar soluções em áreas ainda não analisadas.

Ao mudar de uma solução v para seu vizinho v', seja pelo movimento de troca ou de realocação, considera-se como regra de proibição a ocorrência do par (J_i, J_{i+1}) de v. Apenas no caso de realocação da tarefa para a última posição da sequência, proíbe-se o par (J_i, J_{i-1}) de v. Para impedir o retorno à solução v, armazena-se em uma matriz T de dimensão $n \times n$, sendo n o número de tarefas, na posição (J_i, J_{i+1}), ou (J_i, J_{i-1}), conforme o caso, o valor dado pela soma da iteração corrente com um tempo p de proibição.

Dessa forma, um elemento é considerado tabu quando o tempo tabu armazenado na matriz é maior que o número da iteração corrente do procedimento Busca Tabu. Esse prazo de proibição p é selecionado aleatoriamente no intervalo [$0{,}9 \times PrazoTABU \leq p \leq 1{,}1 \times PrazoTABU$] , sendo *PrazoTABU* um parâmetro do método.

A estratégia adotada para paralelizar a Busca Tabu consiste em acionar um novo *thread* para refinar uma solução sempre que houver melhora na solução global, mantendo-se a Busca Tabu em sua execução normal. Assim, são executados em paralelo o procedimento Busca Tabu e um ou mais processos de refinamento. Desta forma, dos $N_{processors}$ núcleos de processamento disponíveis, um é reservado para a Busca Tabu e os demais para os procedimentos de refinamento. É adotada uma restrição de no máximo $N_{processors}$ - 1 processos de refinamento estarem em execução simultânea com a Busca Tabu. O objetivo é evitar que um número excessivo de *threads* estejam concorrendo entre si, o que aumentaria o tempo gasto pelo sistema operacional com a comutação de contextos dos processos para agendamento do novo *thread* para execução (SILBERSCHATZ *et al.*, 2004). Além disso, não há garantia de que substituindo-se um processo de refinamento por outro haja melhora no procedimento.

O pseudocódigo do procedimento Busca Tabu é mostrado no Algoritmo 4. Nesse pseudocódigo, $N_{processors}$ indica a quantidade de núcleos de processamento disponíveis, *PrazoTABU* o tempo tabu, T a matriz tabu, *difElite* e *CE* são parâmetros utilizados para a

construção do Conjunto Elite que é utilizado no método Reconexão por Caminhos, *TABUmax* é o número máximo de iterações tabu sem melhora e *MRDmax* o número máximo de iterações do método de busca local MRD. O procedimento retorna a melhor solução, representada por *v**.

Algoritmo 4: TABU-M

Entrada: $N_{processors}$, v^*, *PrazoTABU*, T, *difElite*, *CE*, *TABUmax*, *MRDmax*
Saída: v^*

1 **início**
2 $v_{\min} \leftarrow v^*$ `// Melhor solução corrente`
3 $v \leftarrow v^*$ `// Solução auxiliar`
4 *IterCorrente* $\leftarrow 0$ `// Iteração corrente`
5 *MelhorIter* $\leftarrow 0$ `// Iteração da melhor solução`
6 $T \leftarrow \emptyset$ `// Lista Tabu`
7 $CE \leftarrow \emptyset$ `// Conjunto Elite`
8 *NumProcsRefinamento* $\leftarrow 0$ `// Contador de processos de refinamento`
9 **enquanto** (*IterCorrente* − *MelhorIter* $\leq$ *TABUmax*) **faça**
10 Seja $v' \leftarrow v \oplus m$ o melhor elemento de $V \subseteq N(v)$, sendo $N = N^T$ quando *IterCorrente* for par e $N = N^R$ se *IterCorrente* for ímpar, e tal que o movimento m não seja tabu, ou se tabu, $f(v') < f(v^*)$
11 Atualize a Matriz Tabu T com o atributo que representa a solução v'
12 $v \leftarrow v'$
13 **se** $f(v) < f(v^*)$ **então**
14 $v_{\min} \leftarrow v$
15 **se** *NumProcsRefinamento* $<$ $N_{processors} - 1$ **então**
16 *NumProcsRefinamento* $\leftarrow$ *NumProcsRefinamento* $+ 1$
17 NOVO-PROCESSO-Refinamento$_1$(v^*, *v*, *MRDmax*)
18 **senão**
19 Entre com o processo de refinamento na fila de espera. Assim que outro processo em execução terminar, realize um sorteio entre os processos da fila, execute o processo sorteado e atualize o contador de processos.
20 **fim**
21 *MelhorIter* $\leftarrow$ *IterCorrente*
22 Atualize o Conjunto Elite *CE*
23 **fim**
`// Verifica se algum processo atualizou` v^*
24 **se** $(f(v^*) < f(v_{\min}))$ **então**
25 $v_{\min} \leftarrow v^*$
26 $T \leftarrow \emptyset$
27 Retorne para a linha 10
28 **fim**
29 *IterCorrente* $\leftarrow$ *IterCorrente* $+ 1$
30 **fim**
31 Aguarde o término de todos os processos de refinamento
32 **Retorne** v^*
33 **fim**

4.5 Reconexão por Caminhos

A Reconexão por Caminhos – PR (GLOVER, 1996) é uma estratégia que faz um balanço entre diversificação e intensificação. Ela consiste em caminhar de uma solução para outra com vistas a encontrar soluções intermediárias melhores nesse caminho. Em cada uma de

suas iterações é adicionado um atributo da solução alvo (ou solução guia) à solução base, de forma que ao final do procedimento, a solução guia seja encontrada.

No algoritmo proposto, a Reconexão é aplicada conforme segue. Durante a exploração do espaço de busca por TS, é formado um conjunto de soluções elite. Esse conjunto elite (CE) é formado por soluções encontradas por TS durante a exploração do espaço de soluções que sejam de boa qualidade e heterogêneas. Para fazer parte de CE, cada solução candidata deve satisfazer a um dos dois seguintes critérios: 1) ser melhor que a melhor das |CE| soluções do conjunto elite; 2) ser melhor que a pior das |CE| soluções do conjunto elite e se diferenciar de todas elas em um determinado percentual dos atributos, definido por *difElite*. Um atributo é definido como alguma tarefa J_i da sequência. Para exemplificar, considere as sequências v_1 = {1, 2, 4, 5, 6, 3, 7, 9, 8, 10} e v_2 = {1, 2, 4, 5, 6, 9, 3, 10, 7, 8}. Elas têm 50% de atributos iguais, a saber, as tarefas J_1=1, J_2=2, J_3=4, J_4=5 e J_5=6. O objetivo desta estratégia é evitar a inclusão em CE de soluções muito parecidas. Estando o conjunto elite já formado e ordenado de forma crescente, isto é, a melhor solução ocupando a primeira posição do conjunto e a pior, a última, quando uma solução entra e o conjunto já está completo, a de pior avaliação sai.

O método PR-M é aplicado ao final do algoritmo GTSPR-M a todos os pares de soluções de CE, de forma bidirecional. Ele funciona como segue. Conhecidas as soluções guia e base, a cada iteração, uma tarefa J_i da solução guia é inserida na mesma posição i na solução base. Para manter a consistência, a tarefa da solução base que foi substituída sai de sua posição e assume a posição daquela que entrou. Ao final das iterações, ou seja, após a solução base receber todos os atributos da solução guia, o atributo inserido que produzir a melhor solução é fixado e sobre esta solução é aplicado uma descida completa tendo por base movimentos de realocação. Dessa forma, retorna-se um ótimo local com relação à vizinhança N^R.

5 Resultados Computacionais

Para a realização dos experimentos computacionais, foi utilizado um conjunto problemas-teste da literatura com 8, 9, 10, 11, 12, 15, 20, 25, 30, 40, 50 e 75 tarefas. Há nesse conjunto 144 problemas-teste no total, sendo 12 problemas para cada número de tarefas.

O algoritmo heurístico GTSPR-M foi desenvolvido com a linguagem C++, utilizando-se o IDE *NetBeans* 6.7 e o compilador GCC 4.3.3. Todos os experimentos computacionais foram executados em um computador *Pentium Core 2 Quad* (Q6600) 2,4 GHz (4 núcleos de processamento) com 4 GB de memória RAM e sistema operacional *Ubuntu 9.04*. Cada problema-teste foi resolvido 20 vezes, cada qual partindo de uma solução inicial diferente.

Os parâmetros do algoritmo foram os mesmos adotados em Souza *et al.* (2008), isto é: Γ = {0; 0,02, 0,04; 0,12; 0,14}, *difElite* = 0,4, *GRASPmax* = 20, *MRDmax* = $7 \times n$, *TABUmax* = $4 \times n$, *PrazoTABU* = $2 \times n$, |CE| = 5 e 4 núcleos de processamento.

Na Tabela 1 são apresentados os resultados da comparação entre os algoritmos GTSPR-M, proposto neste trabalho, e os algoritmos AGA1, de Ribeiro *et al.* (2009) e GTSPR, de Souza *et al.* (2008). Nessa tabela, a coluna imp^{X} indica quanto o algoritmo GTSPR-M foi superior às melhores soluções produzidas pelo algoritmo X que está sendo comparado, isto é, AGA1 ou GTSPR. Para tal avaliação, foi utilizada a Equação (2), em que f_i^{X*} e $f_i^{\text{GTSPR-M}*}$ representam as melhores soluções do algoritmo X e GTSPR-M, respectivamente. Também é apresentado o desvio médio das soluções encontradas por cada algoritmo X em relação às melhores soluções conhecidas (Equação (3)). Para cada problema-teste i do grupo, f_i^* e $\bar{f}_i^{X}$ representam o melhor valor conhecido e o valor médio encontrados por cada algoritmo X, respectivamente. Por fim, também é mostrado o tempo médio de processamento, em segundos.

$$imp_i^{\text{AGA}} = \frac{f_i^{\text{X}*} - f_i^{\text{GTSPR-M}^*}}{f_i^{\text{X}*}} \qquad (2)$$

$$gap_i^{\text{avg}} = \frac{\bar{f}_i^{\text{X}} - f_i^*}{f_i^*} \qquad (3)$$

Tabela 1: Comparação de resultados GTSPR-M × AGA1 × GTSPR

#Tar.	Tempo (s)			Imp^X (%)		$\overline{gap}^{avg}$ (%)		
	GTSPR-M	AGA1	GTSPR	AGA1	GTSPR	GTSPR-M	AGA1	GTSPR
8	0,05	0,84	0,06	0,00	0,00	0,00	0,00	0,00
9	0,07	1,14	0,09	0,00	0,00	0,00	0,15	0,00
10	0,11	1,44	0,15	0,00	0,00	0,00	0,24	0,00
11	0,17	1,99	0,25	0,00	0,00	0,00	0,03	0,00
12	0,24	2,53	0,37	0,00	0,00	0,00	0,07	0,00
15	0,49	5,42	1,13	0,00	0,00	0,32	0,76	0,47
20	1,71	18,54	4,93	0,00	0,00	0,25	0,73	0,64
25	5,23	41,14	14,9	0,00	0,00	0,84	1,02	1,09
30	14,6	100,86	39,93	0,00	0,00	1,30	1,60	1,68
40	56,36	302,29	190,61	0,04	0,12	2,43	2,45	3,37
50	208,14	806,49	630,77	-0,30	0,02	3,95	4,08	4,95
75	1922,23	1804,54	6308,74	0,88	0,72	6,16	7,76	7,6
Avg	**184,12**	**257,27**	**599,33**	**0,05**	**0,07**	**1,27**	**1,57**	**1,65**

Ressalta-se que na Tabela 1, os resultados do algoritmo GTSPR foram obtidos na mesma máquina utilizada para testar o GTSPR-M. No entanto, os resultados do AGA1 foram obtidos em um Pentium Core 2 Duo 2,1 GHz, com 4 GB de memória RAM, sob plataforma Windows Vista. Assim, para uma comparação mais justa, os tempos do AGA1 foram reduzidos em 10%. Essa redução foi realizada tendo em vista os resultados da comparação de desempenho dos processadores envolvidos, conforme informações disponíveis nos sítios http://www.spec.org/cpu2006/results/res2007q3/cpu2006-20070723-01550.html e http://www.spec.org/cpu2006/results/res2007q1/cpu2006-20070205-00388.html.

Na comparação do GTSPR-M com sua versão sequencial GTSPR, nota-se clara superioridade do algoritmo proposto, conforme se depreende pela Tabela 1. Essa superioridade é verificada tanto com relação à capacidade de produzir soluções melhores, quanto com relação à variabilidade das soluções finais e tempo de processamento.

Na comparação do GTSPR-M com o algoritmo AGA1, verifica-se que o algoritmo proposto é capaz de produzir soluções melhores nos problemas-teste envolvendo 40 e 75 tarefas, mas nesse quesito tem desempenho inferior em problemas de 50 tarefas. Entretanto, considerando todas as instâncias, o desempenho global médio é superior. Na comparação com a capacidade de produzir soluções finais com menor variabilidade, a vantagem do GTSPR-M é clara, principalmente nas instâncias com menor número de tarefas (nos problemas-teste de até 12 tarefas, a variabilidade foi nula). Com relação ao esforço computacional despendido, o GTSPR-M exigiu um tempo de processamento significativamente menor que o AGA1. A exceção ocorreu para os problemas-teste envolvendo 75 tarefas, nos quais ele requereu cerca de 120 segundos a mais, na média. Destaca-se, entretanto, que para esse conjunto de problemas-teste, o AGA1 utilizou um conjunto especial de parâmetros visando a redução de seu tempo de processamento. Observa-se, ainda, que com esse tempo a mais o GTSPR-M conseguiu gerar soluções melhores e com menor variabilidade.

A Figura 1 exibe a relação entre os piores desvios das soluções finais com os desvios médios dessas soluções no primeiro conjunto de problemas-teste. Para cada problema-teste com *n* tarefas, o pior *gap* é dado como o maior desvio dentre todas as instâncias e o *gap* médio como a média dos desvios.

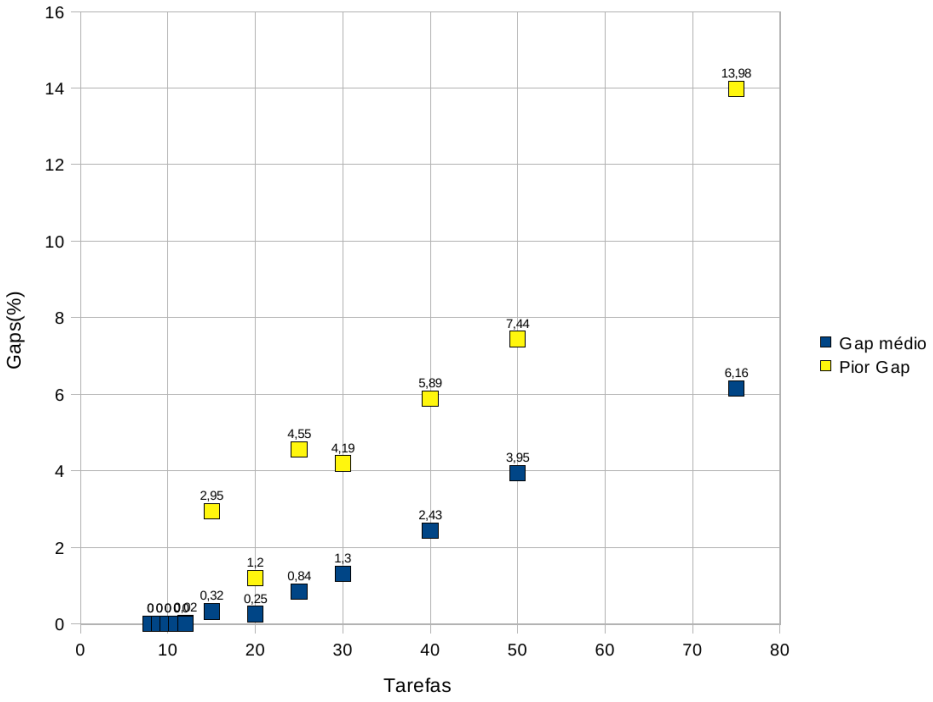

Figura 1: *Gap* Médio × Pior *Gap*

Nota-se, pela Figura 1, que para problemas com até 12 tarefas, ambos os desvios foram nulos. Nos problemas com mais de 12 tarefas, a maior diferença entre o pior *gap* e o *gap* médio foi de 7% para problemas com 75 tarefas. Para os demais problemas, a diferença entre os desvios permaneceu abaixo de 3,5%.

Como nem todos os algoritmos não foram executados na mesma máquina, foi aplicado um teste de probabilidade empírica, seguindo a metodologia de Aiex *et al.* (2002), para verificar a capacidade dos algoritmos alcançarem um valor alvo. Para execução dos experimentos, utilizou-se um problema-teste com 40 tarefas, tendo como alvo a melhor solução conhecida. O algoritmo foi executado 100 vezes, sendo que em cada rodada ele era interrompido após alcançar o alvo. Não foram permitidos tempos de execução repetidos; assim, os tempos repetidos foram descartados e uma nova execução foi feita. Após determinados os tempos para as 100 execuções, estes foram ordenados de forma crescente e, para cada tempo t_i, foi associada uma probabilidade $p_i = (i - 0{,}05)/100$. Os resultados do gráfico $t_i \times p_i$ são apresentados na Figura 2. Nessa figura, são apresentados, também, o comportamento dos algoritmos GILS-VND-RT, GTSPR e GPV de Gomes Jr. *et al.* (2007), Souza *et al.* (2008) e Rosa *et al.* (2009), respectivamente.

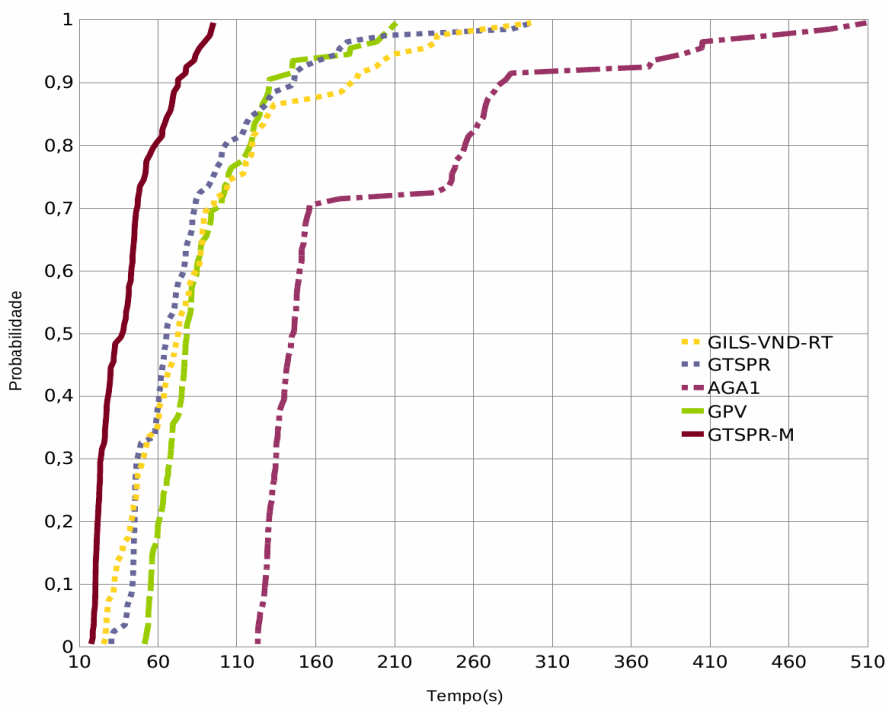

Figura 2: Teste de Probabilidade Empírica

Observa-se na Figura 2 que o algoritmo GTSPR-M demandou menos tempo que todos os outros algoritmos para garantir a melhor solução. De acordo com o gráfico, ele necessita de aproximadamente 95 segundos para retornar a melhor a solução, enquanto que os algoritmos GPV, GILS-VND-RT, GTSPR e AGA1 necessitam, respectivamente, de cerca de 210, 310, 310 e 510 segundos. Verifica-se, também, que os algoritmos GPV, GILS-VND-RT e GTSPR-M obtiveram um desempenho semelhante nos instantes iniciais. O algoritmo AGA1 foi o que demandou mais tempo para alcançar o valor alvo.

6 Conclusões e trabalhos futuros

Este trabalho teve seu foco na resolução do problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção, considerando janelas de entrega e tempo de preparação da máquina dependente da sequência de produção.

Foi proposto um algoritmo paralelo híbrido de três fases, denominado GTSPR-M, que combina os procedimentos heurísticos GRASP e Descida em Vizinhança Variável para a geração de solução inicial, Busca Tabu para refinamento dessa solução, e Reconexão por Caminhos, como mecanismo de pós-otimização.

O algoritmo GTSPR-M foi desenvolvido utilizando-se *threads* para a sua paralelização. Os experimentos computacionais foram executados em uma arquitetura SMP (*Symmetric Multiprocessing*) com um processador de 4 núcleos de processamento. O algoritmo foi testado em um conjunto de problemas-teste da literatura e os resultados obtidos foram comparados com aqueles produzidos por outros algoritmos da literatura, incluindo a versão sequencial do algoritmo proposto. Os resultados obtidos mostraram que o algoritmo proposto se mostrou mais eficiente que os demais algoritmos da literatura. Isso se deveu ao fato de que o GTSPR-M teve capacidade de apresentar as melhores soluções na maioria dos casos e com a menor variabilidade, além de requerer menos tempo de processamento.

Como trabalhos futuros, aponta-se a implementação de uma nova estratégia de

paralelização da fase Busca Tabu, haja vista que essa é a fase que requer a maior parcela do tempo de processamento do GTSPR-M. A proposta a ser implementada é a de paralelizar a vizinhança, isto é, distribuir a análise da vizinhança entre os diversos núcleos de processamento.

Agradecimentos

Os autores agradecem à FAPEMIG (processos CEX-PPM 357/09 e CEX 01201/09) pelo apoio ao desenvolvimento do presente trabalho.

Referências

Aiex, R. M.; Resende, M. G. C. e Ribeiro, C. C. (2002). Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, v. 8, p. 343–373.

Glover, F. (1996). *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, Capítulo Tabu search and adaptive memory programming: Advances, applications and challenges, p. 1–75. Kluwer Academic Publishers.

Glover, F. e Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Boston.

Gomes Jr., A. C.; Carvalho, C. R. V.; Munhoz, P. L. A. e Souza, M. J. F. (2007). Um método heurístico híbrido para a resolução do problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. *Anais do XXXIX Simpósio Brasileiro de Pesquisa Operacional – SBPO*, p. 1649–1660, Fortaleza.

Mladenović, N. e Hansen, P. (1997). Variable Neighborhood Search. *Computers and Operations Research*, v. 24, p. 1097–1100.

Resende, M. G. C. e Ribeiro, C. C. (2005). *Parallel metaheuristics: a new class of algorithms*, Capítulo 14, *Parallel Greedy Randomized Adaptive Search Procedures*, p. 315–344. John Wiley and Sons.

Resende, M. G. C. e Ribeiro, C. C. (2010). GRASP. In: Burke, E. K., Kendall, G. (Eds.), *Search Methodologies*, 2ª ed., Springer (to appear), available at: http://www.ic.u.br/~celso/artigos/grasp.pdf.

Ribeiro, F. F.; Souza, M. J. F. e Souza, S. R. (2009). An adaptive genetic algorithm for solving the single machine scheduling problem with earliness and tardiness penalties. *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics - IEEE SMC 2009*. San Antonio, USA, p. 698–703.

Rosa, B. F. (2009). Heurísticas para o problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. Dissertação de mestrado, Progr. de Pós-Graduação em Modelagem Matemática e Computacional, CEFET-MG, Belo Horizonte.

Rosa, B. F.; Souza, S. R. e Souza, M. J. F. (2009). Formulações de programação matemática para o problema de sequenciamento em uma máquina com janelas de entrega distintas e tempo de preparação dependente da sequência de produção. Anais do XXXII CNMAC. Cuiabá: UFMT, 2009. v.2. p.930 – 936.

Silberschatz, A.; Galvin, P. B. e Gagne, G. (2004). *Fundamentos de Sistemas Operacionais*. LTC, Rio de Janeiro, 6ª edição.

Souza, M. J. F; Ochi, L. S.; Gonçalves, F. A. C. A e Penna, P. H. V. (2008). GRASP, Tabu Search and Path Relinking for solving total earliness/tardiness single machine scheduling problem with distinct due windows and sequence-dependent setups. *Proceedings of the XXIX CILAMCE*, v. 1, 15 p. (CD-ROM), Maceió.

Wan, G. e Yen, B. P. C. (2002). Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. *European Journal of Operational Research*, v. 142, p. 271–281.