SEQUENCIAMENTO EM UMA MÁQUINA: OTIMIZAÇÃO HEURÍSTICA VIA MULTIPROCESSAMENTO PARALELO

Frederico Augusto de Cezar Almeida Gonçalves[1], **Marcone Jamilson Freitas Souza**[2]

[1]Programa de Modelagem Matemática e Computacional,
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Av. Amazonas, 7675, CEP 30510-000, Belo Horizonte (MG), Brasil

[2]Departamento de Computação – Universidade Federal de Ouro Preto (UFOP)
Campus Universitário, Morro do Cruzeiro, CEP 35.400-000, Ouro Preto (MG), Brasil

zepfred@gmail.com, marcone@iceb.ufop.br

Resumo. *Este artigo tem seu foco em uma classe de problemas de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. No problema tratado, 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, denominado GTSPR-M. Esse algoritmo combina os procedimentos heurísticos GRASP, Descida em Vizinhança Variável, Busca Tabu e Reconexão por Caminhos. Para cada sequência gerada pela heurística é utilizado um algoritmo de tempo polinomial para determinar a data ótima de início de processamento de cada tarefa. Os resultados encontrados mostram a superioridade do algoritmo proposto em relação a vários algoritmos da literatura, tanto com relação à qualidade da solução final quanto em relação ao desvio médio e tempo de processamento.*

PALAVRAS-CHAVE*: Sequenciamento em uma máquina, GRASP, Descida em Vizinhança Variável, Busca Tabu, Reconexão por Caminhos, Processamento Paralelo, Threads*

Abstract. *This paper has its focus on a class of single-machine scheduling problems of minimizing total weighted earliness and tardiness penalties. In the problem considered there are sequence-dependent setup times and due windows for each job. In order to solve it, a parallel heuristic algorithm, so-called GTSPR-M, is proposed. This algorithm combines GRASP, Variable Neighborhood Descent, Tabu Search and Path Relinking heuristic procedures. For each solution generated by GTSPR-M, an optimal timing algorithm was used to determine the optimal initial processing dates for each job. The results encountered show that GTSPR-M outperforms others algorithms from literature, in relation to the quality of the final solutions as well as the average deviations and processing times.*

KEYWORDS*: Single-machine scheduling, GRASP, Variable Neighborhood Descent, Tabu Search, Path Relinking, Parallel processing, Threads*

1 INTRODUÇÃO

Neste trabalho é estudada uma classe de problemas de sequenciamento em uma máquina com minimização das penalidades por antecipação e atraso da produção (*Single Machine Scheduling for Minimizing Earliness and Tardiness Penalties*), ou simplesmente PSUMAA. Trata-se aqui o PSUMAA com janelas de entrega distintas e tempos de preparação da máquina dependentes da sequência da produção, ou simplesmente PSUMAA-JP.

O 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 reduzem 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 por meio de heurísticas. Gomes Jr. et al. (2007) propuseram o algoritmo GILS-VND-RT, que combina GRASP (Feo e Resende, 1995), ILS (Lourenço et al., 2003) e VND (Mladenović e Hansen, 1997). O algoritmo explora o espaço de soluções usando movimentos de realocação e troca de tarefas. Além disso, os autores desenvolveram uma formulação de programação matemática que validam os resultados heurísticos em problemas-teste de até 12 tarefas.

Souza et al. (2008) desenvolveram um algoritmo híbrido de três fases, denominado GTSPR, baseado em GRASP, VND, Busca Tabu e Reconexão por Caminhos para o PSUMAA-JP. Na primeira fase foi utilizado 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, um conjunto de soluções denominado conjunto elite é formado. Na terceira e última fase, aplica-se um procedimento de Reconexão por Caminhos aos pares de soluções do conjunto elite como estratégia de pós-otimização. O algoritmo proposto explora o espaço de soluções usando movimentos de troca e realocação de uma tarefa e de um bloco de tarefas. Os resultados obtidos indicam a superioridade desse algoritmo frente ao de Gomes Jr. et al. (2007).

Ribeiro et al. (2010) propuseram um Algoritmo Genético Adaptativo (AGA) para a resolução do PSUMAA-JP. A população inicial é formada pela aplicação da fase de construção GRASP, tendo como função guia várias regras de despacho. Para cada indivíduo é utilizado o algoritmo 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. Os operadores de cruzamento melhor sucedidos em gerações passadas recebem maior probabilidade de escolha. O AGA também aplica periodicamente a Reconexão por Caminhos visando a encontrar soluções intermediárias de melhor qualidade.

Rosa (2009) desenvolveu duas formulações de programação linear inteira mista para o PSUMAA-JP. A primeira delas representa um aperfeiçoamento do modelo de Gomes Jr. et al. (2007), e a outra, uma formulação indexada no tempo. Foi proposto, também, um algoritmo heurístico de duas fases baseado em GRASP, VND e no princípio da otimalidade próxima. Segundo o autor, a formulação indexada no tempo possibilitou a obtenção

de um maior número de soluções ótimas, bem como um menor esforço computacional gasto na resolução dos problemas. Já o algoritmo heurístico, nomeado GPV, obteve tempos computacionais inferiores aos resultados da literatura e se mostrou competitivo com os algoritmos existentes até então.

Todos esses algoritmos investigados não exploram paralelismo. Esse fato motivou a realização da presente pesquisa, já que os computadores atuais dispõem de recursos *multicore*, o que possibilita melhorar o desempenho desses algoritmos.

O presente trabalho, então, trata de cobrir esta lacuna. Apresenta-se um algoritmo heurístico paralelo para resolver o PSUMAA-JP. Em linhas gerais, ele funciona como segue. Na primeira fase, aplica-se GRASP para gerar uma solução inicial. Nesta 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 soluções denominado Conjunto Elite. Este conjunto se caracteriza por possuir boas soluções e diferenciadas entre si. O Conjunto Elite 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 o problema em questão é caracterizado. A Seção 3 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, bem como apresentam-se possíveis propostas a serem exploradas em trabalhos futuros.

2 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 desta tarefa i é delimitado pela janela de entrega $[E_i, T_i]$, onde E_i e T_i indicam respectivamente a data inicial e 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. Não há custo algum se as tarefas forem 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; (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.

3 METODOLOGIA

3.1 Representação da Solução

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. Dessa maneira, na sequência $v = \{4, 3, 5, 6, 2, 1\}$, a tarefa 4 é a primeira a ser realizada e a tarefa 1, a última.

3.2 Vizinhança

A exploração do espaço de soluções é feita utilizando-se 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 $2 \leq k \leq n - 2$. Esses movimentos definem, respectivamente, as estruturas de vizinhança N^T, N^R e N^{Or}.

3.3 Função de Avaliação

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 e t_i $(e_i, t_i \geq 0)$ indicam, 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 e, assim, calcular a quantidade de antecipação e atraso de cada tarefa da sequência.

3.4 Algoritmo GTSPR-M

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

Algoritmo 1: GTSPR-M

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

Na primeira fase (linha 3), constrói-se uma solução inicial. A seguir (linha 4), a solução resultante desta fase é refinada por um algoritmo baseado em Busca Tabu. Como pós-otimização (linha 5) é aplicada a Reconexão por Caminhos. Cada uma dessas fases é detalhada a seguir.

4 Geração da Solução Inicial

A solução inicial é gerada pelo procedimento GRASP-M, descrito no Algoritmo 2. Esse método usa a heurística GRASP (Feo e Resende, 1995), tendo como método de busca local a Descida em Vizinhança variável – VND (Mladenović e Hansen, 1997). Além disso,

Algoritmo 2: GRASP-M

```
   Entrada: N_processors, Γ, GRASPmax, MRDmax
   Saída: v*
1  início
2      NewGRASPmax ← GRASPmax/N_processors
3      f* ← ∞
4      para i=1 até N_processors faça
5          v_i ← NOVO-PROCESSO-GRASP(Γ, NewGRASPmax, MRDmax)
6      fim
7      Aguarde o término de todos os processos GRASP
8      para i=1 até N_processors faça
9          se (f(v_i) < f*) então
10             f* ← f(v_i);  v* ← v_i
11         fim
12     fim
13     Retorne v*
14 fim
```

aplica-se uma estratégia de distribuição do processamento, adotada para que o procedimento possa ser executado em modo paralelo e, assim, tirar proveitos de uma arquitetura com vários núcleos de processamento.

No Algoritmo 2, $N_{processors}$ indica a quantidade de núcleos de processamento disponíveis, *GRASPmax* é o número máximo de iterações GRASP e *MRDmax* o número máximo de iterações do método de busca local MRD (Método Randômico de Descida). O procedimento retorna a melhor solução, v^*, construída.

Como pode ser visto no Algoritmo 2, linha 2, um novo número máximo de iterações para o GRASP é calculado. Este parâmetro *NewGRASPmax* é dependente da quantidade processadores disponíveis ($N_{processors}$) e seu é repassado para o procedimento NOVO-PROCESSO-GRASP (vide linha 5). Após a criação de $N_{processors}$ novos processos GRASP, estes são executados em paralelo e o valor da melhor solução v^* é atualizado com o menor valor encontrado por algum dos processos disparados. Esta estratégia de distribuição de processamento visa melhorar o desempenho com relação à métrica tempo de resposta. O processamento antes executado *GRASPmax* vezes é dividido em $N_{processors}$ núcleos de processamento, o novo número de iterações é menor e os núcleos de processamento que poderiam estar ociosos também são utilizados para construir a solução inicial. A distribuição do processamento não implica em perda de qualidade da solução final, uma vez que esta divisão de processamento na verdade se trata de um balanceamento do número iterações executadas (*GRASPmax*) entre os núcleos de processamento(*NewGRASPmax*). De fato, todas as *GRASPmax* iterações GRASP serão executadas, mas elas serão distribuídas da forma mais equânime possível entre os $N_{processors}$ núcleos de processamento, reduzindo, assim, o tempo de processamento total. 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 valores $\gamma \in [0, 1]$ que definem o tamanho da Lista Restrita de Candidatos (LRC) 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 ja-

nela 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
```

No 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. Mas 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. Mas, se novamente 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. Neste processo, o procedimento Refinamento$_2$ explora o espaço de soluções através 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, retorna-se um ótimo local com relação às vizinhanças N^R, N^T e N^{Or}.

4.1 Busca Tabu

O procedimento Busca Tabu (Glover e Laguna, 1997), cujo pseudocódigo é mostrado no Algoritmo 4, inicia sua execução (linha 4 do Algoritmo 1) a partir da solução

retornada pelo procedimento GRASP-M.

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 $v_{\min} \leftarrow v^*$; $T \leftarrow \emptyset$; Retorne para a linha 10
26 **fim**
27 *IterCorrente* $\leftarrow$ *IterCorrente* $+ 1$
28 **fim**
29 Aguarde o término de todos os processos de refinamento
30 **Retorne** v^*
31 **fim**

No Algoritmo 4, $N_{processors}$ indica o número de núcleos de processamento disponíveis; *PrazoTABU*, o tempo tabu; T, a matriz tabu; *TABUmax*, o número máximo de iterações tabu sem melhora; *MRDmax*, o número máximo de iterações do método de busca local MRD; e *difElite* e *CE* são parâmetros utilizados para a construção do Conjunto Elite. Retorna-se a melhor solução, representada por v^*.

A cada iteração, o espaço de soluções é explorado alternando-se entre as vizinhanças N^T e N^R, nesta ordem. Assim, na primeira iteração são aplicados movimentos de troca, na segunda movimentos de realocação e assim por diante. Após todos os vizinhos serem analisados, é aceito o melhor vizinho que não seja tabu, ou se tabu, que tenha valor da função objetivo menor que o da solução global. Isso é, considera-se a condição de aspiração por objetivo global. O vizinho escolhido é, então, aceito como a solução corrente.

Ao mover-se da solução v para a solução vizinha v', utiliza-se uma Matriz Tabu T, de dimensão $n \times n$, com n representando o número de tarefas, para armazenar a lista de atributos proibidos. Se o vizinho for escolhido pela troca de duas tarefas J_i e J_j $(i < j)$, o atributo que caracteriza este vizinho é definido pelo par de tarefas (J_i, J_{i+1}) de v. Caso o vizinho seja escolhido pela realocação progressiva de uma tarefa J_i para a posição j com $i < j$, o atributo que caracteriza a solução também será representado pelo par de tarefas (J_i, J_{i+1}) de v. Entretanto, para a realocação regressiva em que $i > j$, dois casos são considerados. No primeiro, selecionada a posição j para onde a tarefa J_i será realocada e considerando $i < n$, então o atributo que representará a solução é dado pelo par de tarefas (J_i, J_{i+1}) de v. Para o segundo caso, a tarefa J_i é a última, isto é, $i = n$; desta maneira, o atributo será representado pelo par de tarefas (J_{i-1}, J_i) de v.

O tempo p em que uma solução permanecerá tabu é selecionado aleatoriamente no intervalo $[0,9 \times \textit{PrazoTABU} \leq p \leq 1,1 \times \textit{PrazoTABU}]$, sendo *PrazoTABU* um parâmetro. Esse valor p é somado ao número da iteração corrente e armazenado na Matriz Tabu. Assim, dado o par de tarefas J_i e J_j, o prazo que este elemento permanecerá tabu é armazenado na posição (J_i, J_j) da matriz T. Assim, um elemento é tabu se o número armazenado nessa posição da matriz T for maior que o número da iteração corrente da Busca Tabu.

Pela estratégia adotada para processamento paralelo deste procedimento aciona-se um novo *thread* para processar o refinamento a cada melhora da melhor solução até então, mantendo-se em paralelo a execução normal da Busca Tabu. Assim, são executados em paralelo o procedimento Busca Tabu e um ou mais processos de refinamento. O número de núcleos de processamento disponíveis é definido por $N_{\textit{processors}}$; sendo um núcleo reservado para a Busca Tabu e os demais para os procedimentos de refinamento. É adotada uma restrição de no máximo $N_{\textit{processors}} - 1$ 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). De fato, não há garantia de que substituindo-se um processo de refinamento por outro haja melhora no procedimento.

4.2 Reconexão por Caminhos

A Reconexão por Caminhos (Glover, 1996) é uma estratégia de intensificação geralmente aplicada como pós-otimização ou refinamento de ótimos locais obtidos durante a busca. Dado um par de soluções, uma delas é definida como solução guia e a outra como solução base. O objetivo é partir da solução base e caminhar em direção à solução guia por meio da adição de atributos da solução guia na solução base.

Durante a exploração do espaço de busca por TS, um conjunto de soluções, denominado Conjunto Elite (CE), é construído. Esse conjunto 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 dessa estratégia é

evitar a inclusão em CE de soluções muito parecidas. Estando o conjunto elite já formado, quando uma solução entra, a de pior avaliação sai.

No algoritmo GTSPR-M, o PR é aplicado, de forma bidirecional, a todos os pares de soluções do CE. 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 da 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 essa solução é aplicada uma busca local (*Best Improvement*) baseada em movimentos de realocação. Dessa forma, retorna-se um ótimo local com relação à vizinhança N^R. A Reconexão por Caminhos é feita pelo procedimento PR-M mostrado no Algoritmo 5. Nesse algoritmo, $N_{processors}$ representa o número de núcleos de processamento e v^*, a melhor solução encontrada até então.

Algoritmo 5: PR-M

Entrada: $N_{processors}$, v^*, *CE*
Saída: v^*
início
  LPR $\leftarrow \emptyset$
  Atualize *LPR* com todos os pares de soluções de *CE*
  Execute NOVO-PROCESSO-PR(v^*, v_i, v_j) para cada par de soluções v_i e v_j de *LPR*
  Aguarde o término de todos os processos PR
  Retorne v^*
fim

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. Esse conjunto envolve 144 problemas, sendo 12 problemas-teste para cada número de tarefas.

O algoritmo GTSPR-M foi desenvolvido na 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*.

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. Cada problema-teste foi resolvido 20 vezes, cada qual partindo de uma solução inicial diferente.

Na Tabela 1 são apresentados os resultados da comparação do algoritmo GTSPR-M com os algoritmos heurísticos GILS-VND-RT, GTSPR, AGA1 e GPV, propostos nos trabalhos de Gomes Jr. et al. (2007), Souza et al. (2008), Ribeiro et al. (2010) e Rosa (2009), respectivamente. Na primeira métrica, dada pela Eq. (2), o objetivo é verificar a variabilidade das soluções finais produzidas pelos algoritmos. Nessa medida, calcula-se, para cada algoritmo *Alg* aplicado a um problema-teste i, o desvio médio gap_i^{avg} das soluções encontradas ($\bar{f}_i^{Alg}$) em relação às melhores soluções conhecidas (f_i^*). Na segunda medida, dada pela Eq. (3), calcula-se o quanto o algoritmo proposto foi capaz de superar a melhor solução produzida por cada um dos algoritmos da literatura.

$$gap_i^{avg} = \frac{\bar{f}_i^{Alg} - f_i^*}{f_i^*} \quad (2)$$

$$imp_i^{best} = \frac{f_i^{Alg^*} - f_i^{GTSPR\text{-}M^*}}{f_i^{Alg}} \quad (3)$$

Tabela 1. Resultados GTSPR-M × GILS-VND-RT × GTSPR × AGA1 × GPV

Tarefas	Tempo(s)					$\overline{gap}^{avg}(\%)$					$imp^{best}(\%)$			
	GTSPR-M	**GILS-VND-RT**	**GTSPR**	**AGA1**	**GPV**	**GTSPR-M**	**GILS-VND-RT**	**GTSPR**	**AGA1**	**GPV**	**GILS-VND-RT**	**GTSPR**	**AGA1**	**GPV**
8	0,05	0,03	0,06	0,84	0,11	0,00	0,03	0,00	0,00	0,00	0,00	0,00	0,00	0,00
9	0,07	0,06	0,09	1,14	0,17	0,00	0,06	0,00	0,15	0,00	0,00	0,00	0,00	0,00
10	0,11	0,10	0,15	1,44	0,24	0,00	0,02	0,00	0,24	0,00	0,00	0,00	0,00	0,00
11	0,17	0,18	0,25	1,99	0,46	0,00	0,12	0,00	0,03	0,06	0,00	0,00	0,00	0,00
12	0,24	0,26	0,37	2,53	0,74	0,00	0,21	0,00	0,07	0,02	0,00	0,00	0,00	0,00
15	0,49	0,84	1,13	5,42	1,47	0,32	1,47	0,47	0,76	0,51	0,00	0,00	0,00	0,00
20	1,71	3,91	4,93	18,54	5,19	0,25	1,65	0,64	0,73	1,49	0,00	0,00	0,00	0,00
25	5,23	11,96	14,9	41,14	14,22	0,84	2,32	1,09	1,02	1,53	0,00	0,00	0,00	0,00
30	14,6	36,06	39,93	100,86	30,84	1,30	3,34	1,68	1,60	2,41	0,19	0,00	0,00	0,04
40	56,36	140,21	190,61	302,29	111,52	2,43	4,70	3,37	2,45	3,93	0,47	0,12	0,04	1,09
50	208,14	443,05	630,77	806,49	309,03	3,95	7,29	4,95	4,08	6,48	1,03	0,02	-0,30	1,31
75	1922,23	1231,27	6308,74	1804,54	787,9	6,16	19,38	7,60	7,76	13,64	6,94	0,72	0,88	5,50
Avg	**184,12**	**155,66**	**599,33**	**257,27**	**105,16**	**1,27**	**3,38**	**1,65**	**1,57**	**2,51**	**0,72**	**0,07**	**0,05**	**0,66**

Pelos resultados apresentados na Tabela 1, observa-se o algoritmo GTSPR-M obteve tempos computacionais menores do que os do algoritmo GILS-VND-RT para instâncias de 15 a 50 tarefas. Na variabilidade média das soluções, o algoritmo proposto foi melhor em todas a instâncias, chegando a melhorias de até 100%. As melhores soluções do GILS-VND-RT foram superadas em 0,72% na média. Numa outra comparação com o algoritmo GILS-VND-RT, observa-se que apenas a primeira fase do GTSPR-M foi capaz de superar o GILS-VND-RT em todas as avaliações aplicadas. Na comparação com a sua versão sequencial, apresentada em Souza et al. (2008), o GTSPR-M se mostrou melhor, tanto em relação aos tempos computacionais, quanto na variabilidade média das soluções em todos os problemas-teste. Esta melhoria resultou em redução do tempo computacional de até 70%. O desvio das soluções finais foi reduzido em até 61% e os valores das melhores soluções do GTSPR foram superados em 0,07%, na média. Comparado com o algoritmo AGA1, proposto em Ribeiro et al. (2010), o GTSPR-M também se mostrou melhor no tempo computacional, variabilidade média das soluções finais e capacidade de encontrar melhores soluções finais. A superioridade na redução dos tempos computacionais chegou a 94%, enquanto que na variabilidade média das soluções finais, essa redução foi de 90%. Na última comparação, foi utilizado o algoritmo GPV, desenvolvido em Rosa (2009). Os resultados mostram que para problemas com até 50 tarefas, o tempo computacional do GTSPR-M foi menor, com redução no tempo de até 68%. Com respeito à variabilidade média das soluções finais e à capacidade de obter soluções finais melhores, o algoritmo proposto também foi melhor. Para problemas com 75 tarefas, o GPV demandou um tempo computacional bem menor. No entanto, a qualidade da solução produzida por ele foi pior em cerca de 5,5% com relação à melhor solução alcançada. Considerando apenas os resultados da fase "GRASP-VND" do GTSPR-M, verifica-se que o algoritmo proposto demanda um tempo menor que o GPV para produzir soluções finais melhores.

Como nem todos os algoritmos foram executados na mesma máquina, foi proposto também para a comparação dos resultados um teste de probabilidade empírica (Aiex et al., 2002). 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 $p_i = \frac{i-0,5}{100}$. Os resultados do gráfico $t_i \times p_i$ são apresentados na Figura 1. Neste gráfico são apresentados, também, o desempenho dos algoritmos GPV, AGA1, GILS-VND-RT e GTSPR.

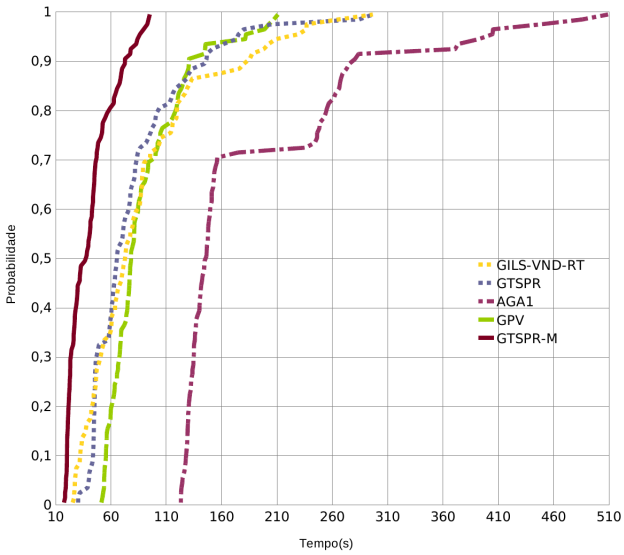

Figura 1. Teste de Probabilidade Empírica

Observa-se na Figura 1 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 alcançar o valor alvo, enquanto que os algoritmos GPV, GILS-VND-RT, GTSPR e AGA1 necessitam, respectivamente, de 210, 310, 310 e 510 segundos. Verifica-se, também, que os algoritmos GPV, GILS-VND-RT e GTSPR 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.

Além disso, o algoritmo GTSPR-M explora a capacidade de multiprocessamento paralelo da geração atual de computadores. Para tanto, a paralelização é feita por meios de *threads*. O algoritmo foi executado em uma máquina dotada de um processador de 4 núcleos operando no Sistema SMP (*Symmetric Multiprocessing*). O algoritmo foi testado em um conjunto de problemas-teste da literatura e os resultados obtidos foram comparados com aqueles produzidos por quatro outros algoritmos da literatura.

Os resultados obtidos mostraram que o algoritmo GTSPR-M se mostrou mais ro-

busto que os demais algoritmos heurísticos da literatura, pois foi o que apresentou as melhores soluções na maioria dos casos e com a menor variabilidade.

Como trabalhos futuros, apontam-se os seguintes: (*i*) Estudar novas estratégias de paralelização dos métodos, de forma a se obter ganhos de desempenho tanto em tempo computacional quanto em qualidade das soluções; (*ii*) Expandir o mecanismo de paralelização para várias máquinas conectadas em uma rede.

Agradecimentos

Os autores agradecem à FAPEMIG (processos CEX-PPM 357/09 e CEX-APQ 01201-09) e FAPERJ (processo E-26/101.023/2007) pelo apoio recebido.

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.

Feo, T. A. e Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133.

Glover, F. (1996). Tabu search and adaptive memory programming: Advances, applications and challenges. Barr, R. S.; Helgason, R. V. e Kennington, J. L., editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, 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.

Lourenço, H. R.; Martin, O. e Stützle, T. (2003). Iterated Local Search. Glover, F. e Kochenberger, G., editors, *Handbook of Metaheuristics*, p. 321–353. Kluwer Academic Publishers.

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 Greedy Randomized Adaptive Search Procedures*. Alba, E., editor, *Parallel metaheuristics: a new class of algorithms*, p. 315–344. John Wiley and Sons.

Ribeiro, F. F.; Souza, S. R.; Souza, M. J. F. e Gomes, R. M. (2010). An adaptive genetic algorithm to solve the single machine scheduling problem with earliness and tardiness penalties. *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, Barcelona.

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

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*, Maceió.