PRODUÇÃO

Produção, v. 22, n. 4, p. 766-777, set./dez. 2012
http://dx.doi.org/10.1590/S0103-65132012005000020

Uma heurística híbrida para minimizar custos com antecipação e atraso do sequenciamento da produção em uma máquina

Puca Huachi Vaz Penna[a*], Marcone Jamilson Freitas Souza[b], Frederico Augusto de Cezar Almeida Gonçalves[c], Luiz Satoru Ochi[d]

[a*]puca@iceb.ufop.br, UFOP, Brasil
[b]marcone@iceb.ufop.br, UFF, Brasil
[c]fred@nti.ufop.br, CEFET-MG, Brasil
[d]satoru@ic.uff.br, UFF, Brasil

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 resolução do problema, desenvolveu-se um algoritmo heurístico de 3 fases, nomeado GTSPR. A primeira fase baseada em GRASP é descida em vizinhança variável para a geração da solução inicial, a segunda fase baseada em busca tabu para refinamento da solução, e por fim a reconexão por caminhos como estratégia de pós-otimização, na terceira fase. 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 computacionais mostraram que o algoritmo GTSPR supera outros algoritmos da literatura, tanto com relação à qualidade da solução final quanto em relação à variabilidade dessas soluções.

Palavras-chave

Sequenciamento em uma máquina. GRASP. Busca tabu. Descida em vizinhança variável. Reconexão por caminhos.

1. Introdução

De acordo com França Filho (2007), o estudo de problemas de programação de tarefas envolvendo penalizações pela antecipação e pelo atraso é mais recente que aquele voltado para problemas em que o objetivo envolve uma função não decrescente do instante de conclusão do processamento da tarefa, tais como tempo médio de fluxo, soma ponderada de atrasos e *makespan* (momento em que termina a execução da última tarefa). Para estes, custos mais elevados decorrem apenas do adiamento da conclusão das tarefas. Entretanto, com a filosofia *just-in-time* adotada por muitas empresas, o foco atual é penalizar também a conclusão das tarefas antes do instante em que elas são requeridas. Isso é justificado pelo fato de que concluir uma tarefa antecipadamente pode resultar em custos financeiros extras pela necessidade antecipada de capital e/ou espaço para armazenamento e/ou de outros recursos para manter e gerenciar o estoque.

Com relação às entregas, são encontrados na literatura 3 tipos de problemas: a) datas de entrega comuns (*common due date*), b) datas de entrega distintas (*distinct due dates*) e c) janelas de entrega distintas (*distinct due windows*).

No primeiro tipo, existe uma única data para entregar todas as tarefas, enquanto no segundo há uma data de entrega específica associada a cada tarefa. No terceiro tipo, há um período para a conclusão de cada tarefa. Segundo Wan e Yen (2002), este último aparece em situações de incertezas ou tolerâncias com relação às datas de entrega e não há custos se as tarefas forem concluídas dentro da janela de entrega.

*UFOP, Ouro Preto, MG, Brasil
Recebido 05/05/2009; Aceito 25/07/2011

Para problemas de produção envolvendo penalizações pela antecipação e pelo atraso com datas comuns de entrega, há muitas propriedades que podem ser consideradas nos algoritmos de solução e que permitem uma redução do esforço computacional na exploração do espaço de busca. Por exemplo, de acordo com Baker e Scudder (1990), se C_i representa a data de conclusão da tarefa *i* e $f(C_i)$, o valor da função custo relativo à tarefa *i*, então na sequência ótima, os pontos $(C_i; f(C_i))$ formam um *V* (diz-se que a sequência é *V-shaped*). Em outras palavras, na sequência ótima, as tarefas antecipadas devem ser ordenadas de forma não crescente pela relação entre o tempo de processamento e a penalização pela antecipação, enquanto as tarefas atrasadas devem ser ordenadas de forma não decrescente pela relação entre o tempo de processamento e a penalização pelo atraso. Outra propriedade interessante é que na sequência ótima não há tempos ociosos entre as tarefas.

Para o caso em que as datas de entrega são distintas, assim como para o caso com janelas de entrega distintas, tais propriedades não são necessariamente válidas, tornando mais complexa a exploração do espaço de busca. Para esses tipos de problemas é necessário saber se é permitida a ociosidade das máquinas ou não. Segundo França Filho (2007), existem situações em que a ociosidade não é possível por gerar custos maiores que aqueles decorrentes da conclusão antecipada das tarefas. Entretanto, há situações em que vale a pena manter uma máquina ociosa, mesmo que haja uma tarefa em condições de ter seu processamento iniciado. Para ilustrar essa situação, sejam duas tarefas *i* e *j*, que devem ser consecutivas em uma sequência de produção envolvendo uma máquina, às quais estão associados os custos unitários por atraso de 100 e 20, respectivamente, custos unitários por antecipação de 22 e 2, e que tenham como datas de entrega os instantes 50 e 55, respectivamente, e 15 e 25 unidades de tempo de processamento, respectivamente. Suponha que a máquina esteja disponível desde o instante 20. Se a tarefa *i* começar no instante 20, será concluída no instante 35 e, portanto, haverá uma penalização igual a 330 (= 15 × 22) pela antecipação. Nesse caso, a tarefa *j* poderá começar somente no instante 35. Considerando seu tempo de processamento, ela será concluída no instante 60, portanto 5 unidades de tempo atrasada em relação à data de entrega, resultando em um custo por atraso igual a 100 (= 5 × 20), e um custo total de 430 (= 330 + 100). Considere, agora, que o processamento da tarefa *i* seja iniciado no instante 35, mesmo sabendo que a máquina está disponível desde o instante 20. Neste caso, a tarefa *i* seria concluída no instante 50, sem multa nem por antecipação nem por atraso. A tarefa *j*, então, já poderia começar, terminando no instante 75, com 20 unidades de tempo de atraso, resultando em uma multa por atraso de 400 (= 20 × 20). O custo total nesta última situação seria de 400 unidades, menor que aquele no qual a tarefa *i* começava desde o instante de liberação da máquina. Assim, não havendo restrições à ociosidade das máquinas, determinar a melhor data para iniciar o processamento de cada tarefa ou, equivalentemente, inserir tempos ociosos entre tarefas poderá produzir soluções de menores custos.

Com relação à influência do tempo de preparação das tarefas na sequência de produção, chamado de tempo de *setup*, observa-se que a maioria dos trabalhos encontrados na literatura considera que tais tempos são negligenciáveis ou podem ser adicionados aos tempos de processamento das tarefas. Isto é, considera-se que os tempos de *setup* são independentes da sequência de tarefas. Contudo, muitos estudos, como os de Panwalkar, Dudek e Smith (1973), apud Gupta e Smith (2006), indicam que tempos de *setup* significativos aparecem com frequência em muitas situações práticas sempre que há alteração da tarefa a ser processada na máquina. Segundo os autores, com base em informações prestadas por gerentes de produção, cerca de 70% dos entrevistados afirmaram que em pelo menos 25% das operações, o tempo de preparação era dependente da sequência de produção.

Dentre os problemas de produção, o de sequenciamento em uma máquina com minimização das penalidades por antecipação e atraso da produção (*single machine scheduling for minimizing earliness and tardiness penalties*), representado simplesmente por PSUMAA, é um dos mais simples.

Além da importância prática desse problema, com inúmeras aplicações industriais (ALLAHVERDI; GUPTA; ALDOWAISAN, 1999), trata-se de um problema difícil de ser resolvido na otimalidade em tempos computacionais aceitáveis, por pertencer à classe NP-difícil (DU; LEUNG, 1990; LIAW, 1999; WAN; YEN, 2002). Essa conjunção de aplicabilidade e dificuldade de resolução na otimalidade tem motivado os pesquisadores a desenvolver algoritmos eficientes para resolvê-lo de forma aproximada.

Neste trabalho, trata-se o PSUMAA com janelas de entrega distintas e tempos de preparação da máquina dependentes da sequência da produção. Apesar de representar uma generalização do PSUMAA, essa variante do PSUMAA tem sido pouco explorada na literatura. De nosso conhecimento, foram encontrados apenas os trabalhos de Gomes Junior et al. (2007) e Ribeiro, De Souza e Souza (2009) e nenhuma publicação em periódico, mesmo com a abrangente revisão de Allahverdi et al. (2008).

Dada sua natureza combinatória, propõe-se resolvê-lo por um algoritmo heurístico de 3 fases, combinando os procedimentos GRASP (FEO; RESENDE, 1995), *variable neighborhood descent* - VND (MLADENOVIC; HANSEN, 1997), busca tabu - *tabu search* (GLOVER, 1986) e reconexão por caminhos - *path relinking* (GLOVER, 1996). Na primeira fase é utilizado um procedimento baseado em GRASP para gerar uma boa solução inicial. Nessa fase, cada solução construída é refinada por um procedimento de busca local baseado em VND. Na segunda fase, a solução inicial é refinada por um procedimento busca tabu (BT). Durante essa fase é formada uma lista de soluções elite (grupo elite), contendo soluções de alta qualidade e diferenciadas entre si. Como pós-otimização, na terceira fase, é aplicada a reconexão por caminhos a pares de soluções do grupo elite.

De acordo ainda com Allahverdi et al. (2008), a meta-heurística busca tabu é uma das mais utilizadas para resolver problemas de sequenciamento. Como essa técnica requer uma solução inicial de qualidade, nada mais natural que utilizar a meta-heurística GRASP para gerá-la. Por sua vez, somente a fase de construção GRASP não guia, em geral, a ótimos locais com relação às estruturas de vizinhanças adotadas. Assim, aplica-se também um método de busca local para melhorar as soluções construídas. A meta-heurística VND foi usada para esse fim, pois requer poucos parâmetros (basicamente a definição das vizinhanças e a ordem de exploração destas), produz soluções de boa qualidade (MLADENOVIC; HANSEN, 1997) e é de fácil implementação. A aplicação de reconexão por caminhos deveu-se ao sucesso desta técnica na solução de vários outros problemas combinatórios (GLOVER; KOCHENBERGER, 2003).

Este trabalho apresenta relevância por tratar de um problema de sequenciamento ainda pouco estudado e contribuir com o desenvolvimento de uma ferramenta computacional para a otimização de processos produtivos, com aplicações potenciais em várias indústrias, como as petroquímicas, siderúrgicas, mineradoras etc.

O restante deste trabalho está estruturado como segue. Na seção 2 é feita uma apresentação de trabalhos correlatos. As características do problema estudado são detalhadas na seção 3, enquanto a seção 4 descreve o algoritmo desenvolvido para resolvê-lo. Na seção 5 são apresentados e discutidos os resultados computacionais. A seção 6 conclui o trabalho e aponta perspectivas futuras.

2. Trabalhos relacionados

Vários trabalhos tratam o PSUMAA com datas de entrega. Gordon, Proth e Chu (2002) apresentam um estudo sobre datas comuns de entrega em problemas de sequenciamento em uma máquina e em máquinas paralelas. De acordo com os autores, a data de entrega se tornou um fator importante a partir de meados da década de 80 devido à filosofia *just-in-time*. Vários métodos de formulação da função objetivo foram apresentados, além de algumas propriedades para datas comuns de entrega.

Coleman (1992) resolveu o PSUMAA com datas de entrega distintas e tempos de preparação dependentes da sequência usando um modelo de programação inteira mista, aplicando-os em problemas-teste com 4 e 8 tarefas.

Lee e Choi (1995) aplicaram algoritmos genéticos (AG) ao PSUMAA com datas de entrega distintas. Para determinar a data ótima de conclusão de processamento de cada tarefa da sequência produzida pelo AG, desenvolveram um algoritmo específico, de complexidade polinomial, que explora as características do problema.

James (1997) resolveu o PSUMAA com data comum de entrega sem tempo ocioso entre as tarefas utilizando a meta-heurística busca tabu. O autor utilizou 3 movimentos para explorar o espaço de busca: troca entre tarefas adjacentes, troca entre duas tarefas quaisquer e inserção de uma tarefa à frente de outra. Experimentos mostraram que a troca adjacente obteve os piores resultados, enquanto um movimento híbrido de troca e inserção teve melhor desempenho.

Rabadi, Mollaghasemi e Anagnostopoulos (2004) focaram no PSUMAA com datas comuns de entrega e tempo de preparação da máquina dependente da sequência. Os autores apresentam um procedimento *branch-and-bound* que é capaz de resolver na otimalidade, em tempos aceitáveis, problemas-teste de até 25 tarefas, o que representava um avanço porque até aquela data algoritmos exatos para essa classe de problemas eram capazes de resolver somente problemas-teste de até 8 tarefas.

Hallah (2007) tratou o PSUMAA com datas de entrega distintas, não sendo permitida a existência de tempo ocioso entre as tarefas. O autor propõe um método híbrido que combina heurísticas de busca local (regras de despacho, método da descida e *simulated annealing*) e um algoritmo evolucionário.

Bilge, Kurtulan e Kirac (2007) resolveram o problema de sequenciamento em uma máquina com penalidades apenas por atraso e datas de entrega distintas por meio da meta-heurística busca tabu (BT). Para gerar a solução inicial os autores utilizaram vários métodos, entre eles as heurísticas *earliest due date* - EDD e *weighted shortest processing time* - WSPT (BAKER; SCUDDER, 1990). A BT desenvolvida usou

uma estrutura de vizinhança híbrida com movimentos de troca e realocação.

Wan e Yen (2002) apresentaram um método baseado na meta-heurística busca tabu (BT) para resolver o PSUMAA com janelas de entrega distintas. Para cada sequência de tarefas gerada pela busca tabu é acionado um procedimento de complexidade polinomial para determinar a data de conclusão ótima do processamento de cada tarefa da sequência. Esse procedimento é adaptado daquele proposto em Lee e Choi (1995), em que se consideram janelas de entrega em lugar de datas de entrega. Os autores exploram o espaço de busca usando movimento de troca de pares de tarefas adjacentes e algumas propriedades do problema.

Gomes Junior (2007) e Gomes Junior et al. (2007) desenvolveram um modelo de programação linear inteira mista para o PSUMAA com janelas de entrega e tempos de preparação dependentes da sequência. O modelo desenvolvido foi utilizado para resolver na otimalidade problemas-teste de até 12 tarefas. Essa modelagem serviu para comparar os resultados obtidos por um algoritmo heurístico baseado em GRASP, *iterated local search* - ILS (GLOVER; KOCHENBERGER, 2003) e VND, também proposto pelos autores. Para cada sequência gerada pela heurística, é acionado um algoritmo para determinar a data ótima de conclusão do processamento de cada tarefa. Tal algoritmo foi adaptado de Wan e Yen (2002) e inclui o tempo de preparação da máquina. O algoritmo desenvolvido foi capaz de encontrar todas as soluções ótimas conhecidas.

Ribeiro, De Souza e Souza (2009) trataram o mesmo problema de Gomes Junior et al. (2007) e desenvolveram um algoritmo genético autoadaptativo, denominado AGA, para resolvê-lo. O AGA parte de uma população inicial gerada pela combinação de procedimentos baseados em heurísticas construtivas gulosas, parcialmente gulosas (fase de construção GRASP - FEO; RESENDE, 1995) e aleatórias. A seguir, o algoritmo evolui pela aplicação de cinco operadores de cruzamento e uma mutação baseada em troca de posição de tarefas. Esses operadores de cruzamento têm, inicialmente, a mesma probabilidade de serem escolhidos. Entretanto, em função do processo evolutivo, os operadores de cruzamento mais bem-sucedidos têm aumentadas as chances de serem escolhidos nas próximas gerações. Esse procedimento de alteração nas chances de escolha dos operadores é inspirado no algoritmo GRASP reativo de Prais e Ribeiro (2000). Periodicamente, o algoritmo aplica o procedimento reconexão por caminhos (GLOVER, 1996), tendo como base o melhor indivíduo gerado ao longo de todo o processo evolutivo e como guia o melhor indivíduo gerado por cada operador nas últimas cinco gerações. Os autores comparam os resultados obtidos pelo AGA com os de Gomes Junior et al. (2007) e concluem que o algoritmo proposto é capaz de encontrar soluções de melhor qualidade e com menor variabilidade.

Uma revisão mais abrangente de trabalhos sobre sequenciamento com tempos de preparação dependentes da sequência pode ser encontrada em Allahverdi, Gupta e Aldowaisan (1999) e Allahverdi et al. (2008).

3. O problema de sequenciamento abordado

O problema de sequenciamento em uma máquina tratado neste trabalho tem as seguintes características:

(i) Uma máquina deve processar um conjunto de n tarefas;

(ii) A cada tarefa i está associado um tempo de processamento P_i e uma janela de entrega $[E_i, T_i]$, indicando uma data inicial E_i e uma data final T_i 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. As tarefas concluídas dentro da janela de entrega não possuem custo de penalização;

(iii) A máquina pode executar no máximo uma tarefa por vez e uma vez iniciado o processamento de uma tarefa não é permitida a sua interrupção;

(iv) Todas as tarefas estão disponíveis para processamento na data 0;

(v) Entre duas tarefas i e j consecutivas é necessário um tempo S_{ij} de preparação (*setup*) da máquina, o qual é dependente da sequência. Assume-se que a máquina não necessita de tempo de preparação inicial;

(vi) É permitido tempo ocioso entre a execução de duas tarefas consecutivas.

A Figura 1 ilustra o sequenciamento de 3 tarefas, em que para cada uma delas são apresentados a janela de entrega $[E_i, T_i]$ e o tempo de preparação entre elas. Como se observa, na sequência apresentada, a tarefa 3 necessita de um tempo ocioso de forma

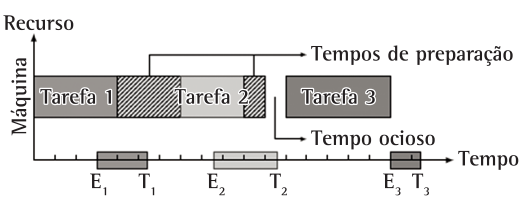

Figura 1. Características do problema estudado.

que seu término coincida com o início de sua janela de entrega para que não haja custo por antecipação.

Seguindo a notação a|b|c de Allahverdi et al. (2008), podemos descrever o problema com a seguinte notação $1|ST_{sd}, E_i \leq d_i \leq T_i|\Sigma^n_{i=1}(\alpha_i e_i + \beta_i t_i)$, onde:

1. Tipo de máquina (campo a):
   - 1, indicando uma máquina (*single machine*).
2. Característica de processamento (campo b):
   - ST_{sd}, representando tempo de preparação dependente da sequência (*sequence-dependent setup time*);
   - $E_i \leq d_i \leq T_i$, indicando janela de entrega distinta (*distinct due window*).
3. Função objetivo (campo c):
   - $\Sigma^n_{i=i}(\alpha_i e_i + \beta_i t_i)$, representando penalização por antecipação e atraso (*total weighted earliness and tardiness*).

4. Metodologia

A utilização de procedimentos heurísticos para resolver o PSUMAA se deve à complexidade combinatória do problema tratado. O algoritmo proposto baseia-se em meta-heurísticas, as quais, ao contrário dos procedimentos heurísticos clássicos, têm mecanismos que possibilitam escapar das armadilhas dos ótimos locais.

4.1. *Algoritmo GTSPR*

O algoritmo proposto, denominado GTSPR, possui 3 fases e combina características dos procedimentos GRASP, *variable neighborhood descent* (VND), busca tabu (*tabu search*) e reconexão por caminhos (*path relinking*). Seu pseudocódigo é apresentado na Figura 2.

Na primeira fase, é construída uma solução inicial utilizando-se GRASP e VND. Na segunda fase essa solução é refinada por um procedimento baseado em busca tabu. Na terceira e última fase, como pós-otimização, é aplicado reconexão por caminhos a pares de soluções elite produzidas durante a busca tabu.

```
Algoritmo GTSPR (v, γ, divElite, |T|, GRASPmax, MRDmax, TSmax)
1  Início
2      v0 ← GRASP - VND (v, γ, GRASPmax, MRDmax)
3      v1 ← TS (v0, |T|, divElite, GE, TSmax, MRDmax)
4      v ← PR (GE, divElite, MRDmax);
5      Retorne v
6  Fim GTSPR
```

Figura 2. Algoritmo GTSPR.

O algoritmo heurístico proposto, após gerar uma sequência de produção, utiliza um procedimento, descrito em Gomes Junior et al. (2007), para programá-la, isto é, determinar as datas ótimas de início e término de cada tarefa na sequência gerada.

4.2. *Representação de uma solução*

Uma solução do PSUMAA é representada por um vetor v de n elementos, onde cada posição $i = 1, 2, ..., n$ indica a ordem de produção da tarefa v_i da sequência. Assim, na sequência $v = \{5, 3, 2, 1, 4, 6\}$, a tarefa 5 é a primeira a ser realizada e a tarefa 6, a última.

4.3. *Vizinhança de uma solução*

Para explorar o espaço de soluções são utilizados 3 tipos de movimentos: troca da ordem de processamento de duas tarefas da sequência de produção, realocação de uma tarefa para outra posição na sequência e realocação de um bloco de k tarefas, $2 \leq k \leq n - 2$. Esses movimentos definem, respectivamente, as estruturas de vizinhança N^T, N^R e N^{Or}.

Dado um conjunto com n tarefas, há $n(n-1)/2$ vizinhos possíveis na vizinhança N^T. Por exemplo, a solução $v' = \{5, 4, 2, 1, 3, 6\}$ é um vizinho da solução v na vizinhança N^T, pois é obtido pela troca da tarefa 3, que está na segunda posição de v, com a tarefa 4, que está na quinta posição de v.

Na segunda estrutura de vizinhança, N^R, há $(n-1)^2$ vizinhos possíveis. Por exemplo, a solução $v' = \{5, 2, 1, 4, 3, 6\}$ é vizinha de v na vizinhança N^R, pois é obtida pela realocação da tarefa 3 para depois da tarefa 4 na sequência de produção v.

Já na terceira estrutura de vizinhança, N^{Or}, há $(n-k-1)^2$ vizinhos possíveis. Por exemplo, a solução $v' = \{2, 1, 5, 3, 4, 6\}$ é vizinha de v na vizinhança N^{Or}, pois é obtida pela realocação do bloco de tarefas 5 e 3, com $k = 2$, para depois da tarefa 1 na sequência de produção v. Nas vizinhanças N^R e N^{Or} são permitidos movimentos para posições sucessoras ou antecessoras àquela em que a tarefa se encontra na sequência de produção.

4.4. *Função de avaliação*

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

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

em que 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.

Para determinar as datas ótimas de início de processamento das tarefas da sequência dada e, consequentemente, determinar os valores e_i e t_i acima, utiliza-se o procedimento ADDOIP de Gomes Junior et al. (2007).

4.5. *Geração da solução inicial*

A solução inicial é gerada por um procedimento que combina GRASP com VND. A Figura 3 esquematiza esse procedimento.

Na fase de construção (linha 5 da Figura 3), uma solução é formada gradativamente, sendo que a cada passo é adicionada uma única tarefa à sequência. Para se escolher a tarefa a ser adicionada é formada uma lista de candidatos (LC) com todas as tarefas ainda não sequenciadas. Essas são ordenadas pela data de início da janela de entrega de cada tarefa, sendo a com a data mais cedo (E_{min}) a primeira da lista e a de data mais tarde (E_{max}) a última, tal como na heurística EDD.

A partir de LC é formada uma lista restrita de candidatos (LRC), com as tarefas mais bem classificadas segundo o critério da data de início da janela de entrega das tarefas. O tamanho dessa lista restrita de candidatos é definido por um parâmetro $\gamma \in [0, 1]$, escolhido em um conjunto Γ. Fazem parte da LRC todas as tarefas *i* cujas datas E_i sejam menores ou iguais a $E_{min} + \gamma \times (E_{max} - E_{min})$. A seguir, é escolhida aleatoriamente uma tarefa dessa lista, sendo a mesma adicionada à solução parcial. A fase de construção é encerrada quando todas as tarefas forem alocadas.

Na fase de refinamento do procedimento GRASP-VND (linha 6 da Figura 3) é aplicado o procedimento VND_1 a cada solução construída. Essa heurística de refinamento explora o espaço de soluções usando as vizinhanças N^R e N^T. Inicialmente é feita uma descida randômica usando-se movimentos de realocação. Para tanto, é escolhida aleatoriamente uma tarefa e uma posição para inseri-la. Se esta nova sequência produzir uma solução com um valor menor para a função de avaliação, a nova sequência é aceita e passa a ser a solução corrente, caso contrário, é testada outra realocação aleatória. A fase de realocação termina quando houver *MRDmax* iterações consecutivas sem melhora na função de avaliação. Nesta última situação, passa-se a fazer movimentos de troca, também de forma aleatória.

Se for produzida uma solução global de melhor qualidade, interrompe-se esta fase e volta-se para a fase de realocação, caso contrário, prossegue-se com as trocas. O procedimento é encerrado quando não houver melhora na solução global nem com movimentos de realocação nem com movimentos de troca.

Deve-se observar que a solução resultante dessa busca não é necessariamente um mínimo local com relação às vizinhanças consideradas, visto que a vizinhança completa não é analisada a cada iteração. Em virtude desse fato, decorridas *GRASPmax* iterações, a melhor das soluções obtidas é submetida a novo refinamento (linha 12 da Figura 3), desta vez pelo procedimento VND_2. Esse procedimento consiste em explorar o espaço de soluções usando-se, além dos movimentos tradicionais de realocação e troca, também o movimento *Or*.

No procedimento VND_2, a busca é feita de acordo com a seguinte estratégia de exploração do espaço de soluções: 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 e 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 um bloco de *k* tarefas, *k* assume inicialmente seu valor mais baixo. Não havendo melhora, *k* é aumentado progressivamente até atingir seu valor máximo. As descidas randômicas são interrompidas quando houver *MRDmax* iterações sem melhora. Observa-se que o procedimento VND_2 retorna um ótimo local com respeito às vizinhanças N^R, N^T e N^{Or}.

```
Procedimento GRASP - VND (v, γ, GRASPmax, MRDmax)
1   Início
2     f* ← ∞
3     Para i = 1 até GRASPmax faça
4       Escolha γ ∈ Γ
5       v ← Constrói_Solução_GRASP (γ)
6       v' ← VND1 (v, MRDmax)
7       se (f (v') < f*) então
8         v* ← v'
9         f* ← f (v')
10      Fim-se
11    Fim-para
12    v ← VND2 (v*, MRDmax)
13    Retorne v
14  Fim GRASP - VND
```

Figura 3. Procedimento GRASP-VND aplicado ao PSUMAA.

4.6. *Busca tabu aplicada ao PSUMAA*

O procedimento busca tabu (*tabu search* - TS) implementado começa sua execução partindo de uma solução inicial gerada pelo procedimento GRASP-VND. A cada iteração todos os vizinhos desta solução são avaliados, sendo escolhido o melhor deles que não seja tabu ou, se tabu, satisfaça a condição de aspiração, no caso, de o vizinho gerar um valor melhor que o da melhor solução até então. A exploração da

vizinhança é feita tendo por base as estruturas de vizinhança N^T e N^R, alternando-as a cada iteração, isto é, na primeira iteração explora-se o melhor vizinho com movimentos de troca, na segunda iteração com movimentos de realocação e assim por diante. O objetivo de se verificar se o movimento é tabu ou não é evitar ciclos de soluções recentemente visitadas.

Quando um vizinho $v' \in N^T(v) \cup N^R(v)$ é escolhido, ele se torna a solução corrente, independentemente de ele ser melhor ou pior que a solução corrente, e um atributo que caracterize essa solução é inserido em uma lista tabu *T*, proibindo o retorno a ele durante um prazo. Dada a troca entre uma tarefa *i* e uma tarefa *j* ($i < j$), o atributo que é inserido na lista tabu é a subsequência (tarefa *i*, tarefa sucessora de *i* da solução *v*). No caso de realocação de uma tarefa *i* para a frente (realocação progressiva), o movimento tabu é a subsequência (tarefa *i*, sucessor da tarefa *i* na solução *v*). Caso a realocação seja para trás (realocação regressiva), o movimento tabu é a subsequência (tarefa antecessora a *i* na solução *v*, tarefa *i*). A lista tabu é limitada a $|T|$ elementos. Quando ela está completa, o elemento mais antigo é retirado da lista, e aquele recentemente obtido é inserido.

Para mostrar o funcionamento da lista tabu, considere a sequência $v = \{5, 3, 2, 1, 4\}$ e um movimento envolvendo a troca da tarefa 5 com a tarefa 1. O resultado é a sequência vizinha $v' = \{1, 3, 2, 5, 4\}$, sendo na lista tabu *T* incluída a subsequência formada por (tarefa 5, tarefa subsequente a 5 na sequência *v*), isto é, $T \leftarrow T \cup (5, 3)$.

Sempre que há melhora na solução global, é aplicado o procedimento VND_2. Caso ele seja bem-sucedido, a lista tabu é inicializada novamente, já que se muda a região do espaço de busca onde a exploração está sendo feita. O procedimento busca tabu é encerrado quando forem realizadas *TSmax* iterações sem melhora na solução global.

O pseudocódigo TS está descrito na Figura 4. A atualização do grupo elite (GE), linhas 11 e 18 da Figura 4, é feita de acordo com a subseção 4.7.

```
Procedimento TS (v, |T|, divElite, GE, TSmax, MRDmax)
1   Início
2     v* ← v
3     Iter ← 0; {Contador do número de iterações}
4     MelhorIter ← 0
5     T ← ∅
6     GE ← ∅
7     Enquanto (Iter - MelhorIter ≤ TSmax) faça
8       Iter ← Iter + 1
9       Seja v' ← v ⊕ m o melhor elemento de V ⊆ N(v) (N=N^T se
        Iter for ímpar e N=N^R se Iter for par), tal que o movimento
        m não seja tabu (m ∉ T') ou, v' se tabu, atenda a condição
        de aspiração f(v') < f(v*)
10      Atualize a lista Tabu T
11      Atualize o grupo elite GE
12      v ← v'
13      se f(v) < f(v*) então
14        v1 ← VND2 (v, MRDmax)
15        se f(v1) < f(v) então
16          T ← ∅
17          v ← v1
18          Atualize grupo elite GE
19        Fim-se
20        v* ← v
21        MelhorIter ← Iter
22      Fim-se
23    Fim-enquanto
24    v ← v*
25    Retorne v
26  Fim TS
```

Figura 4. Procedimento TS aplicado ao PSUMAA.

4.7. *Reconexão por caminhos*

Reconexão por caminhos (*path relinking* - PR), proposto por Glover (1996), é uma estratégia que faz um balanço entre intensificação e diversificação. É utilizada ou como pós-otimização de uma solução ou como refinamento de ótimos locais obtidos durante a busca. Dado um par de soluções, o objetivo da técnica é partir de uma delas, dita solução base, e chegar à outra, dita solução guia, por meio da adição gradativa na solução base de atributos da solução guia.

No algoritmo proposto, a reconexão por caminhos é utilizada como ferramenta de pós-refinamento aplicada após a execução da fase TS. Para tanto, durante a exploração do espaço de busca pelo procedimento TS é gerado um conjunto de soluções, conhecido como grupo elite (GE), que é utilizado pelo procedimento PR. Para fazer parte desse grupo, cada solução candidata deve satisfazer a um desses dois critérios: a) ser melhor que a melhor das |GE| soluções do grupo elite e b) ser melhor que a pior das |GE| soluções do grupo elite e se diferenciar delas por determinado percentual dos atributos, dado por *divElite*.

O atributo considerado é o par de tarefas *i* e *j* consecutivas. Assim, por exemplo, dadas as sequências $v_1 = \{1, 4, 3, 2, 5, 6\}$ e $v_2 = \{5, 3, 2, 1, 4, 6\}$, elas têm 40% de atributos iguais, a saber, as subsequências (1, 4) e (3, 2). O objetivo dessa estratégia é evitar a inclusão de soluções muito similares no grupo elite. Estando esse grupo já formado, quando uma solução entra, a de pior avaliação sai.

Na implementação do procedimento PR para cada par de soluções elite caminha-se de forma bidirecional, isto é, tanto da pior solução para a melhor (dita reconexão por caminhos progressiva - *forward path relinking*), quanto da melhor para a pior solução (dita reconexão por caminhos regressiva - *backward path relinking*).

Determinadas as soluções base e guia, o procedimento é executado inserindo-se gradativamente um atributo da solução guia na solução base. Por exemplo, dada a solução guia = {5, 3, 2, 1, 4, 6}, a cada iteração insere-se um par de tarefas consecutivas desta solução, a saber (5, 3), (3, 2), (2, 1), (1, 4), (4,

6), na solução base. Cada par é inserido na ordem em que aparece na solução guia. Após cada inserção, as tarefas da solução base que foram substituídas saem da posição que ocupavam e assumem as posições daquelas que entraram. A solução base sofre, então, uma busca local, no caso, uma descida randômica usando os movimentos de troca e realocação, onde se fixam os atributos da solução guia que já foram incorporados à solução base. A cada iteração, a solução base recebe todos os atributos ainda não incorporados da solução guia, um de cada vez, gerando uma nova sequência para cada atributo específico da solução guia. Ao final da iteração, o atributo inserido que produziu a melhor solução após a busca local é incorporado em definitivo à solução base, se tornando fixo. O procedimento se encerra quando a solução guia é alcançada, isto é, quando a solução base passa a ter todos os atributos da solução guia.

Para ilustrar o funcionamento do procedimento, é mostrado, a seguir, um exemplo no qual a solução guia é a sequência {5, 3, 2, 1, 4, 6} e a solução base é {2, 6, 5, 1, 4, 3}.

O objetivo é avaliar a inserção, na solução base, de cada um dos 5 pares de subsequências existentes da solução guia, {(5, 3), (3, 2), (2, 1), (1, 4), (4, 6)}, na ordem em que aparecem e verificar qual inserção traz o melhor resultado segundo a função de avaliação. Para tanto, procede-se como segue:

Passo 1) Inserir as tarefas 5 e 3 do par (5, 3) nas duas primeiras posições da solução base. Para tanto, a tarefa 5 ocupará a primeira posição da solução base, onde está a tarefa 2. Esta, por sua vez, sairá da primeira posição e irá para a terceira posição, onde está a tarefa 5, resultando na sequência {5, 6, 2, 1, 4, 3}. A seguir, é inserida a tarefa 3 na segunda posição da sequência base. Como nesta posição está a tarefa 6, esta se deslocará para a última posição, onde está localizada a tarefa 3. Após esta inserção obtém-se a sequência {5, 3, 2, 1, 4, 6}.

Passo 2) Inserir as tarefas 3 e 2 do par (3, 2) na segunda e terceira posição, respectivamente, da solução base. Para tanto, a tarefa 3 passará a ocupar a segunda posição da solução base, onde está a tarefa 6. Esta, por sua vez, sairá da segunda posição e irá para a última, onde está a tarefa 3, resultando na sequência {2, 3, 5, 1, 4, 6}. A seguir, é inserida a tarefa 2 na terceira posição da sequência base. Como nesta posição está a tarefa 5, esta se deslocará para a primeira posição, onde está localizada a tarefa 2. Após esta inserção obtém-se {5, 3, 2, 1, 4, 6}.

E assim sucessivamente até o passo 5, onde devem-se inserir as tarefas 4 e 6 da última subsequência (4, 6) na quinta e sexta posição, respectivamente, da solução base e prosseguir como nos passos 1 e 2.

Uma estratégia de busca local foi considerada durante a reconexão por caminhos. A cada inserção faz-se uma busca local com o procedimento VND_3 descrito adiante, e a inserção que produzir o melhor resultado para a função de avaliação, dentre os passos 1 a 5, é realizada. O procedimento VND_3 consiste em uma descida randômica usando movimentos de troca seguida de descida completa com movimentos de realocação. Nesse procedimento, sempre que a descida completa com movimentos de realocação produzir uma solução de melhora, retorna-se à descida randômica com movimentos de troca. Durante a aplicação do VND_3 é proibida qualquer movimentação das tarefas da solução guia que foram inseridas. Por exemplo, se a sequência do passo 1 resultar em melhor valor para a função de avaliação, as tarefas 5 e 3 não podem movimentar-se. Assim, ter-se-á inserido na solução base um atributo da solução guia. A inserção dos demais atributos segue um procedimento semelhante, adicionando-se o fato de que, durante a busca local, não somente o par recentemente inserido é fixado, mas todos os demais pares de subsequências da solução guia já inseridos também permanecem fixos.

Toda vez que o procedimento PR gera uma solução melhor que a melhor do grupo elite, a inserção de atributos é interrompida, o grupo elite é atualizado e o processo é reiniciado com a aplicação da reconexão por caminhos envolvendo a melhor solução e as demais do grupo elite. Se encerradas as inserções dos atributos o procedimento tiver gerado uma solução melhor que a pior, satisfazendo os critérios de diversidade estabelecidos anteriormente, o grupo elite é atualizado com a saída de seu pior elemento. Nesse caso, também aplica-se a reconexão por caminhos envolvendo a solução alterada e as demais do grupo elite. O procedimento termina após a reconexão ser aplicada a todos os pares de solução elite e não houver alteração na melhor solução. Neste caso, retorna-se à melhor solução encontrada, bem como conjunto elite.

5. Resultados computacionais

Para testar o algoritmo GTSPR foram utilizados os problemas-teste de Gomes Junior et al. (2007). Essa base de dados foi criada por esses autores a partir de parâmetros adotados em Wan e Yen (2002). Ela envolve um número de tarefas igual a 8, 9, 10, 11, 12, 15, 20, 25, 30, 40, 50 e 75, totalizando 144 problemas-teste no total, sendo 12 para cada número de tarefas.

O algoritmo proposto foi desenvolvido na linguagem C++, utilizando o compilador GCC 4.4.3. Os experimentos foram realizados em um computador *Intel Core i7* 2,93 GHz, 4GB de memória e sistema

operacional *Linux Ubuntu* 10.04. Cada problema-teste foi executado 30 vezes. Os parâmetros do algoritmo são apresentados na Tabela 1, onde *n* é o número de tarefas.

Os resultados encontrados nos experimentos computacionais estão sintetizados nas Tabelas 2 e 3. Na Tabela 2, a primeira coluna indica o número de tarefas do problema, e a cada conjunto de 3 colunas é apresentada a contribuição de cada fase do algoritmo GTSPR no esforço de minimização da função de avaliação *f*. A coluna "fase GRASP-VND" diz respeito à fase de obtenção da solução inicial, a coluna "fase TS" à fase de refinamento por busca tabu e a coluna "fase PR" à fase de pós-otimização por reconexão por caminhos. A coluna "tempo" diz respeito ao tempo médio de processamento acumulado, em segundos.

Para cada fase do algoritmo são aplicadas duas medidas de desempenho, dadas pelas Equações 2 e 3. Nessas equações, para cada problema-teste *i* do grupo, $f_i^{GTSPR^*}$ representa o melhor valor encontrado pelo algoritmo proposto, $f_i^{BKS^*}$ é o melhor valor da literatura (obtido por GOMES JUNIOR et al., 2007 e/ou por RIBEIRO; DE SOUZA; SOUZA 2009), f_i^* é o valor da melhor solução conhecida e $\bar{f}_i^{GTSPR}$ é o valor médio encontrado pelo algoritmo proposto nas diversas execuções para o *i*-ésimo problema-teste.

$$imp_i^{best} = \frac{f_i^{BKS^*} - f_i^{GTSPR^*}}{f_i^{BKS^*}} \quad (2)$$

$$gap_i^{avg} = \frac{\bar{f}_i^{GTSPR} - f_i^*}{f_i^*} \quad (3)$$

Na primeira medida de desempenho, verifica-se o quanto o algoritmo proposto superou os melhores resultados de Gomes Junior et al. (2007) e/ou Ribeiro, De Souza e Souza (2009). Para tanto, para cada problema-teste, aplica-se a Equação 2 e apresenta-se na Tabela 2, para cada fase do algoritmo, a média percentual desses valores para cada grupo de problemas-teste com mesmo número de tarefas.

Na segunda medida de desempenho, é utilizada a Equação 3 para calcular o desvio das soluções médias do algoritmo em relação às melhores soluções existentes. Nessa medida, quanto menor o desvio (*gap*), menor a variabilidade das soluções finais, e mais robusto é o algoritmo.

Tabela 1. Parâmetros do algoritmo GTSPR.

Parâmetro	Valores
Γ	{0; 0,02; 0,04; 0,12; 0,14}
GRASPmax	20
MRDmax	$7 \times n$
TSmax	$4 \times n$
$\|T\|$	$2 \times n$
$\|GE\|$	5
divElite	0,4

Pela Tabela 2 é possível observar que, para problemas com até 25 tarefas, apenas a fase GRASP-VND do algoritmo é suficiente para encontrar as melhores soluções, com um desvio médio de até 2,71%. Nesses problemas-teste, o desvio médio é reduzido na fase TS, ficando em até 1,35%, e a variabilidade das soluções finais é ainda reduzida para 1,17%, na média, com a fase PR.

Para problemas com 30 a 50 tarefas, observa-se que a fase GRASP-VND retorna resultados piores que os encontrados na literatura. Entretanto, na fase TS as melhores soluções da literatura para os problemas-teste de 30 tarefas são encontradas, enquanto que para os demais problemas-teste o algoritmo proposto consegue superar os resultados existentes na média em até 0,73%, com um desvio médio máximo de 5,46%, sendo reduzido para no máximo 5,18% na fase PR. Por fim, os problemas-teste envolvendo 75 tarefas apresentaram melhoras significativas já na primeira fase de 1,78%, passando a 5,38% na terceira fase. Com relação ao desvio médio para este conjunto de problemas, a variabilidade das soluções finais reduziu de 10,06% na fase GRASP-VND para 6,57% na fase TS e para 6,41% na fase PR.

A Tabela 3 apresenta uma comparação entre o algoritmo GTSPR e os outros dois algoritmos encontrados na literatura para o problema, o GILS-VND-RT de Gomes Junior et al. (2007) e o AGA de Ribeiro, De Souza e Souza (2009). Essa comparação é feita em termos do desvio das soluções médias dos algoritmos e do desvio da melhor solução. O desvio da melhor solução é obtido utilizando-se a Equação (3), substituindo-se o valor médio encontrado $\bar{f}_i^{GTSPR}$ pelo melhor valor encontrado, definido por $f_i^{GTSPR^*}$. Em relação aos tempos computacionais uma comparação exata é difícil, pois os algoritmos foram executados em máquinas diferentes. Gomes Junior et al. (2007) utilizaram um PC Athlon XP 3000+ para executar o GILS-VND-RT, enquanto Ribeiro, De Souza e Souza (2009) executaram os testes em um PC *Intel Core* 2 Duo 2,1 GHz. No entanto, como uma tentativa de comparação aproximada, foi utilizada a lista de equipamentos disponibilizada por Dongarra (2010), em que o autor apresenta a velocidade, em *Millions of Floating-Point Operations per Second* (Mflops/s), de diversos computadores. Dessa forma foi assumida a velocidade de 3.176 Mflops/s para o Athlon XP 3000+ e 4.579 Mflops/s para o Core 2 Duo 2,1 GHz. A velocidade desses dois computadores foi adaptada baseada em equipamentos similares, pois não se encontrava na lista. Para o equipamento utilizado

Tabela 2. Resultados computacionais por fase do GTSPR.

Nº. tarefas	Fase GRASP-VND			Fase TS			Fase PR		
	$\overline{imp}^{best}$ (%)	$\overline{gap}^{avg}$ (%)	Tempo[1] (segundos)	$\overline{imp}^{best}$ (%)	$\overline{gap}^{avg}$ (%)	Tempo[1] (segundos)	$\overline{imp}^{best}$ (%)	$\overline{gap}^{avg}$ (%)	Tempo[1] (segundos)
8	0,00	0,00	0,02	0,00	0,00	0,02	0,00	0,00	0,03
9	0,00	0,00	0,02	0,00	0,00	0,03	0,00	0,00	0,04
10	0,00	0,00	0,04	0,00	0,00	0,05	0,00	0,00	0,06
11	0,00	0,00	0,05	0,00	0,00	0,07	0,00	0,00	0,10
12	0,00	0,06	0,07	0,00	0,01	0,10	0,00	0,00	0,14
15	0,00	1,03	0,19	0,00	0,67	0,31	0,00	0,58	0,41
20	0,00	2,02	0,55	0,00	0,81	1,07	0,00	0,74	1,41
25	0,00	2,71	1,32	0,00	1,35	3,17	0,00	1,17	3,98
30	–0,42	4,81	3,17	0,00	2,39	8,70	0,00	2,16	10,51
40	–1,02	7,64	9,88	0,08	3,79	35,26	0,08	3,50	40,77
50	–2,56	9,68	25,53	0,73	5,46	118,74	0,73	5,18	134,16
75	1,78	10,06	169,64	5,31	6,57	1194,84	5,38	6,41	1290,50
Média	–0,18	3,15	17,54	0,51	1,75	113,53	0,52	1,65	123,51

[1]PC Intel Core i7 2,93 GHz.

Tabela 3. Desvio das soluções obtidas por GILS-VND-RT, AGA e GTSPR.

Nº. tarefas	GILS-VND-RT Gomes Junior et al. (2007)			AGA Ribeiro et al. (2009)			GTSPR		
	$\overline{gap}^{best}$ (%)	$\overline{gap}^{avg}$ (%)	Tempo[(1)] (segundos)	$\overline{gap}^{best}$ (%)	$\overline{gap}^{avg}$ (%)	Tempo[(2)] (segundos)	$\overline{gap}^{best}$ (%)	$\overline{gap}^{avg}$ (%)	Tempo[(3)] (segundos)
8	0,00	0,03	0,04	0,00	0,00	0,94	0,00	0,00	0,03
9	0,00	0,06	0,07	0,00	0,14	1,26	0,00	0,00	0,04
10	0,00	0,02	0,11	0,00	0,23	1,60	0,00	0,00	0,06
11	0,00	0,12	0,20	0,00	0,03	2,21	0,00	0,00	0,10
12	0,00	0,21	0,29	0,00	0,07	2,81	0,00	0,00	0,14
15	0,00	1,42	0,94	0,00	0,71	6,02	0,00	0,58	0,41
20	0,00	1,61	4,35	0,00	0,71	20,60	0,00	0,74	1,41
25	0,00	2,18	13,29	0,00	0,98	45,72	0,00	1,17	3,98
30	0,19	3,19	40,07	0,00	1,55	112,06	0,00	2,16	10,51
40	0,41	4,32	155,79	–0,02	2,30	335,88	0,03	3,46	40,77
50	1,22	6,57	492,28	0,19	4,15	885,72	0,03	5,16	134,16
75	6,04	14,51	1368,08[(4)]	4,01	10,07	2002,71[(4)]	–5,38	6,38	1290,50
Média	0,66	2,85	17,54	0,35	1,75	284,79	–0,44	1,64	123,51
Tempo médio corrigido			94,08[(5)] (32,98[(6)])			223,34[(5)] (100,87[(6)])			123,51 (17,42[(6)])

[(1)]PC Athlon XP 64 Bits 3000+; [(2)]PC Intel Core 2 Duo 2,1 GHz; [(3)]PC Intel Core i7 2,93 GHz; [(4)]Tempo obtido utilizando uma configuração diferente do algoritmo original; [(5)]Tempo médio corrigido utilizando-se a métrica Mflops/s (DONGARRA, 2010); [(6)]Tempo médio corrigido sem levar em consideração os problemas-teste com 75 tarefas.

na execução do GTSPR, o *Intel Core* i7 2,93 GHz, foi executado o programa desenvolvido por Dongarra (2010) e obtida a velocidade de 5.839 Mflops/s.

Pela Tabela 3 pode-se observar que o GTSPR obteve um resultado superior em termos do desvio médio, tanto em relação ao desvio da melhor solução quanto em relação ao desvio da solução média. O GTSPR foi o único algoritmo que conseguiu obter, na média, soluções melhores que as melhores soluções conhecidas, sendo essa melhora de 0,44%. Por outro lado, o GILS-VND-RT obteve soluções piores em 0,66% na média e o AGA em 0,35%. Pela comparação dos desvios médios das soluções é possível observar que o GTSPR obteve um desvio médio de apenas 1,64% contra 2,85% do GILS-VND-RT e de 1,75% do AGA.

Ao se comparar os valores absolutos dos tempos computacionais dos algoritmos, pode-se observar que GTSPR demandou menor tempo de processamento para produzir soluções com qualidade melhor ou igual àquelas geradas pelos demais algoritmos. Entretanto, ao se analisar os tempos médios corrigidos, pode-se observar que o algoritmo GILS-VND-RT foi o que demandou menor esforço computacional, apenas 94,08 segundos contra 123,51 segundos do GTSPR e 223,34 segundos do AGA. Como Gomes Junior et al. (2007) e Ribeiro, De Souza e Souza (2009) aplicaram

seus algoritmos aos problemas-teste envolvendo 75 tarefas com uma parametrização diferente, com o objetivo de reduzir o tempo de processamento, uma análise dos tempos computacionais envolvendo apenas os problemas-teste com tarefas entre 8 e 50 foi efetuada. Nessa avaliação o GTSPR demandou o menor tempo computacional, gastando apenas 17,42 segundos, cerca da metade do tempo gasto pelo GILS-VND-RT (que exigiu 32,98 segundos) e menos de 20% do tempo requerido pelo AGA, que exigiu 100,87 segundos.

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 (PSUMAA), considerando janelas de entrega e tempo de preparação da máquina dependente da sequência de produção.

Foi proposto um algoritmo de 3 fases, denominado GTSPR, que combina os procedimentos GRASP e descida em vizinhança variável (*variable neighborhood descent* - VND), para geração de solução inicial, busca tabu (*tabu search* - TS) para refinamento e reconexão por caminhos (*path relinking* - PR), como mecanismo de pós-otimização. Para cada sequência gerada pelo algoritmo são determinadas as datas ótimas de conclusão das tarefas por meio de um procedimento de tempo polinomial da literatura. Para explorar o espaço de soluções são utilizados 3 tipos de movimentos, baseados em troca entre tarefas, realocação de uma tarefa para outra posição e realocação de um bloco de tarefas para outra posição.

O algoritmo GTSPR foi aplicado a problemas-teste envolvendo até 75 tarefas e comparado com dois algoritmos da literatura. Os experimentos computacionais mostraram que, para problemas-teste de até 25 tarefas, apenas a fase GRASP-VND do algoritmo foi suficiente para alcançar as melhores soluções existentes. Todas as demais fases do algoritmo tiveram sua contribuição para a melhora da qualidade da solução final, reduzindo sua variabilidade. Com o algoritmo proposto foi possível superar os resultados da literatura em até 5,38%, com uma variabilidade média de 6,41%, no máximo. Em termos de tempo computacional, o algoritmo proposto também foi competitivo, sendo capaz de produzir soluções melhores em menor tempo de processamento em praticamente todos os problemas-teste.

Como trabalhos futuros, indicam-se os seguintes aperfeiçoamentos: 1) Aplicar periodicamente a estratégia de reconexão por caminhos durante a exploração da busca, e não somente como mecanismo de pós-otimização e 2) Estudar propriedades que possam ser aplicadas ao problema, de forma a reduzir o espaço de busca e, consequentemente, o tempo de processamento do algoritmo.

Referências

ALLAHVERDI, A.; GUPTA, J. N. D.; ALDOWAISAN, T. A review of scheduling research involving setup considerations. *OMEGA*, v. 27, p. 219-239, 1999. http://dx.doi.org/10.1016/S0305-0483(98)00042-5

ALLAHVERDI, A. et al. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, v. 187, n. 3, p. 985-1032, 2008. http://dx.doi.org/10.1016/j.ejor.2006.06.060

BAKER, K. R.; SCUDDER, G. D. Sequencing with Earliness and Tardiness Penalties: A Review. *Operations Research*, v. 38, p. 22-36, 1990. http://dx.doi.org/10.1287/opre.38.1.22

BILGE, U.; KURTULAN, M.; KIRAC, F. A tabu search algorithm for the single machine total weighted tardiness problem. *European Journal of Operational Research*, v. 176, p. 1423-1435, 2007. http://dx.doi.org/10.1016/j.ejor.2005.10.030

COLEMAN, B. J. A simple model for optimizing the single machine early/tardy problem with sequence-dependent setups. *Production and Operation Management*, v. 1, n. 2, p. 225-228, 1992. http://dx.doi.org/10.1111/j.1937-5956.1992.tb00354.x

DONGARRA, J. J. *Performance of various computers using standard linear equations software*. Computer Science Department, University of Tennessee, 2010. Technical Report CS-89-85.

DU, J.; LEUNG, J. Y. T. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, v. 15, p. 483-495, 1990. http://dx.doi.org/10.1287/moor.15.3.483

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109-133, 1995. http://dx.doi.org/10.1007/BF01096763

FRANÇA FILHO, M. F. *GRASP e Busca Tabu aplicados a problemas de programação de tarefas em máquinas paralelas*. 2007. Tese (Doutorado em Engenharia de Sistemas)-Universidade Estadual de Campinas, Campinas, 2007.

GLOVER, F. Future paths for Integer Programming and links to Artificial Intelligence. *Computers and Operations Research*, v. 5, p. 553-549, 1986.

GLOVER, F. Tabu search and adaptive memory programming - Advances, applications and challenges. In: BARR, R. S.; HELGASON, R. V.; KENNINGTON, J. L. (Eds.). *Computing Tools for Modeling, Optimization and Simulation*: Interfaces in Computer Science and Operations Research. Kluwer Academic Publishers, 1996. p. 1-75.

GLOVER, F.; KOCHENBERGER, G. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

GOMES JUNIOR, A. C. *Problema de sequenciamento de uma máquina com penalidades por antecipação e atraso da produção*: modelagem e resolução. 2007. Dissertação (Mestrado em Engenharia de Produção)-Universidade Federal de Minas Gerais, Belo Horizonte, 2007.

GOMES JUNIOR, A. C. et al. Um método heurístico híbrido para a resolução do problema de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL - SBPO, 39.; 2007, Fortaleza. *Anais*... Fortaleza, 2007. p. 1649-1660.

GORDON, V.; PROTH, J.; CHU, C. A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, v. 139, p. 1-25, 2002. http://dx.doi.org/10.1016/S0377-2217(01)00181-3

GUPTA, S. R.; SMITH, J. S. Algorithms for single machine total tardiness scheduling with sequence dependent setups, *European Journal of Operational Research*, v. 175, p. 722-739, 2006. http://dx.doi.org/10.1016/j.ejor.2005.05.018

HALLAH, R. M. Minimizing total earliness and tardiness on a single machine using a hybrid heuristic, *Computers and Operations Research*, v. 34, p. 3126-3142, 2007. http://dx.doi.org/10.1016/j.cor.2005.11.021

JAMES, R. J. W. Using tabu search to solve the common due date early/tardy machine scheduling problem. *Computers and Operations Research*, v. 24, n. 3, p. 199-208, 1997. http://dx.doi.org/10.1016/S0305-0548(96)00052-4

LEE, C. Y.; CHOI, J. Y. A Genetic Algorithm for Job Sequencing Problems with Distinct Due Dates and General Early-Tardy Penalty Weights. *Computers and Operations Research*, v. 22, p. 857-869, 1995. http://dx.doi.org/10.1016/0305-0548(94)00073-H

LIAW, C. F. A Branch-and-Bound Algorithm for the Single Machine Earliness and Tardiness Scheduling Problem. *Computers and Operations Research*, v. 26, p. 679-693, 1999. http://dx.doi.org/10.1016/S0305-0548(98)00081-1

MLADENOVIC, N.; HANSEN, P. Variable Neighborhood Search, *Computers and Operations Research*, v. 24, p. 1097-1100, 1997. http://dx.doi.org/10.1016/S0305-0548(97)00031-2

PANWALKAR, S. S.; DUDEK, R. A.; SMITH, M. L. Sequencing research and the industrial scheduling problem. In: ELMAGHRABY, S. E. (Eds.). *Symposium on the Theory of Scheduling and its Applications*. Springer: Berlin, 1973. p. 29-38.

PRAIS, M.; RIBEIRO, C. C. An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS - Journal on Computing*, v. 12, p. 164-176,2000.http://dx.doi.org/10.1287/ijoc.12.3.164.12639

RABADI, G.; MOLLAGHASEMI, M.; ANAGNOSTOPOULOS, G. C. A Branch-and-Bound Algorithm for the Early/Tardy Machine Scheduling Problem with a Common due-date and Sequence-Dependent Setup Time. *Computers and Operations Research*, v. 31, p. 1727-1751, 2004. http://dx.doi.org/10.1016/S0305-0548(03)00117-5

RIBEIRO, F. F.; DE SOUZA, S. R.; SOUZA, M. J. F. An adaptive genetic algorithm for solving the single machine scheduling problem with earliness and tardiness penalties. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS - IEE SMC, 2009, San Antonio. Proceedings... San Antonio, 2009. p. 698- 703.

WAN, G.; YEN, B. P. C. 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, 2002. http://dx.doi.org/10.1016/S0377-2217(01)00302-2

Agradecimentos

Os autores agradecem à FAPERJ, CNPq e FAPEMIG pelo apoio ao desenvolvimento do presente trabalho, bem como aos revisores anônimos pelos valiosos comentários, que guiaram a uma versão melhorada deste trabalho.

A hybrid heuristic algorithm for job scheduling problem on a single-machine

Abstract

This paper deals with the single-machine scheduling problem with earliness and tardiness penalties. Sequence dependent setup times and distinct due windows are considered. In order to solve this problem, a three-phase heuristic approach, the so-called GTSPR, was developed. The first phase is based on GRASP and Variable Neighborhood Descent to generate an initial solution; the second phase is based on Tabu Search for solution refining, finally, Path Relinking is used as a mechanism of post-optimization. For each job sequence generated by the heuristic, an optimal timing algorithm is used to determine the completion time for each job in the job sequence. Computational experiments carried out show that GTSPR outperforms the previous algorithms found in the related literature, regarding the quality of the final solution and the average gap.

Keywords

Single-Machine. GRASP. Tabu search. Variable neighborhood descent. Path relinking.