Um Algoritmo GRASP para o Seqüenciamento de Tarefas em uma Máquina com Atraso Total Ponderado

José E. C. Arroyo, Fabrício L. S. Silva, Alexandre F. Araújo

Departamento de Informática, Universidade Federal de Viçosa
Campus Universitário da UFV, 36570-00 Centro Viçosa, MG, Brasil
jarroyo@dpi.ufv.br, facol@dpi.ufv.br, alexandre.araujo@ufv.br

Abstract. Neste artigo é abordado o problema de seqüenciamento de tarefas em uma máquina minimizando o atraso total ponderado. Este problema pertence à classe NP-Difícil e é um problema importante existente na programação da produção. O método utilizado para resolver o problema é baseado na heurística de busca GRASP que, além das fases construtiva e busca local, possui uma fase de intensificação baseada da técnica *Path Relinking*. A heurística proposta é testada em problemas disponíveis na OR-*Library*. Os resultados obtidos mostram que a heurística GRASP é um método eficiente para resolver o problema abordado.

Keywords: heurísticas, otimização combinatorial, scheduling.

1 Introdução

O problema de seqüenciamento ou escalonamento de tarefas em uma máquina com o objetivo de minimizar o atraso total ponderado é denotado por SMTWT (do inglês "*single machine total weighted tardiness*"). O problema SMTWT consiste em determinar a seqüência de processamento de n tarefas independentes sobre uma máquina. A máquina processa uma tarefa por vez e sem interrupções. Cada tarefa i possui, um tempo de processamento p_i, uma data de entrega d_i e uma prioridade de entrega w_i. Dada uma seqüência de processamento das tarefas, para cada tarefa i são calculados o instante de finalização C_i (*completion time*) e o atraso da tarefa $T_i = \max\{C_i - d_i, 0\}$. O Atraso total ponderado é determinado por $T_w = \sum_{i=1}^{n} w_i T_i$.

O problema SMTWT ocorre freqüentemente nas indústrias de manufatura onde é necessário administrar os recursos/equipamentos, seqüenciar e processar os pedidos no chão de fábrica e atender aos prazos de entrega dos produtos produzidos.

O problema SMTWT é NP-difícil ([9], [18], [19]), instâncias com mais de 50 tarefas não podem ser resolvidos otimamente por algoritmos exatos tais como *Branch-and-Bound* e Programação Dinâmica [1].

Na literatura sobre o problema SMTWT, os métodos mais usados são baseados em heurísticas e metaheurísticas. As heurísticas construtivas usam regras de despacho para construir uma solução (seqüência de tarefas) fixando uma tarefa em uma posição a cada passo. Em [5], [13] e [21] são propostas algumas heurísticas construtivas. Estas

heurísticas são bastante rápidas, no entanto a qualidade de soluções encontradas não é muito boa quando comparação às soluções ótimas locais obtidas por métodos metaheurísticos e técnicas de busca local [22].

As metaheurísticas são também algoritmos rápidos e consistem em melhorar repetidamente uma solução utilizando técnicas de busca local. Mesmo não garantindo otimalidade global, as metaheurísticas podem encontrar uma grande quantidade de ótimos locais de excelente qualidade. Dentre os algoritmos metaheurísticos propostos para o problema SMTWT podem ser citados: busca tabu ([4], [7]) *simulated annealing* ([20], [22]), algoritmo genético ([7], [11]) e *ant colony optimization* [8]. Congram, Potts e Van de Velde (2002) [6] desenvolvem um método iterativo bastante eficiente que é baseado na combinação de Programação Dinâmica e técnicas de busca local (denominado *Iterated Dynasearch*).

Na biblioteca virtual OR-Library [3] são disponibilizados instâncias do problema SMTWT e as melhores soluções obtidas pelos algoritmos em [6] e [7].

Neste artigo é proposto um algoritmo metaheurístico GRASP (*Greedy Randomized Adaptive Search Procedure*) que utiliza um procedimento de intensificação baseada na técnica *Path Relinking* [17]. Esta técnica de intensificação consiste em explorar trajetórias que conectem duas soluções de elite, a primeira denominada "*origem*" e a segunda denominada "*guia*".

O desempenho do algoritmo proposto neste trabalho é avaliado através da comparação com os melhores resultados, para o problema SMTWT, disponíveis na literatura. Vale ressaltar que, na literatura, não foram encontradas aplicações da metaheurística GRASP para o problema SMTWT. No entanto, esta metaheurística já foi aplicada para resolver outros tipos de problemas de escalonamento de tarefas em uma máquina obtendo excelentes resultados ([2], [16]).

Este artigo é organizado da seguinte maneira. Na Seção 2 são apresentados os detalhes da implementação do algoritmo híbrido GRASP-PR proposto. Os testes realizados e resultados computacionais são mostrados na Seção 3. A Seção 4 apresenta as conclusões do trabalho.

2 Algoritmo GRASP Proposto

GRASP é uma heurística iterativa de múltiplas partidas, proposto inicialmente por Feo e Resende, (1995) [10]. Cada iteração da heurística consiste de duas fases: construção e busca local. A primeira fase constrói uma solução s_1 viável utilizando um algoritmo guloso aleatorizado. Na segunda fase a solução construída s_1 é melhorada através de um método de Busca Local ou busca em vizinhança. A melhor solução obtida s^*, durante todas as iterações GRASP, é retornada. Na Figura 1 apresenta-se um pseudocódigo do algoritmo GRASP básico. O procedimento GRASP-B tem dois parâmetros de entrada, o número máximo de iterações (I_{max}) e a taxa de aleatoriedade (α) usada na fase de construção. O algoritmo GRASP tem sido aplicado com muito sucesso para resolver diversos problemas de otimização combinatória [12].

```
Procedimento GRASP-B ( I_max , α)
1.  f(s*) ← ∞ (infinito);
2.  para i ← 1 até I_max faça
3.    s1 ← ConstruçãoGulosaAleatorizada(α);
4.    s2 ← BuscaLocal(s1 );
5.    se f(s2) < f(s*) então
6.      s* ← s2;
7.  fim for
8.  Retorne s*;
fim GRASP
```

Fig. 1. GRASP básico.

Neste artigo, é proposta uma adaptação da metaheurística GRASP para resolver o problema SMTWT. A metaheurística, além das fases de Construção de Busca Local, possui uma fase de Intensificação, que consiste em combinar a solução s_2 obtida pela Busca Local com a melhor solução s^* encontrada até o momento. Esta combinação é feita através da técnica *Path Relinking* proposta originalmente por Glover (1996) [15]. Na Figura 2 apresentam-se os passos (em forma de pseudocódigo) da metaheurística proposta que é denotada por GRASP-PR. Note que a intensificação com *Path Relinking* não é executada na primeira iteração ($i = 1$), pois a melhor solução s^* só existirá a partir da segunda iteração.

```
Procedimento GRASP-PR ( I_max, α)
1.  f(s*) ← ∞ (infinito);
2.  para i ← 1 até I_max faça
3.      s1 ← ConstruçãoGulosaAleatorizada(α);
4.      s2 ← BuscaLocal(s1 );
5.      se i ≥ 2 então
6.          s3 ← PathRelinking( s2, s*);
7.          se f(s3) < f(s*) então  s* ← s3;
8.      senão
9.          se f(s2) < f(s*) então s* ← s2;
10. fim for
11. Retorne s*;
fim GRASP-PR
```

Fig. 2. GRASP com *Path Relinking*.

Nas subseções seguintes, são detalhadas cada uma das etapas do algoritmo GRASP-PS aplicado para resolver o problema SMTWT onde é minimizado o atraso total ponderado.

2.1 Construção Gulosa Aleatorizada

Nesta etapa é construída uma solução inicial utilizando uma estratégia gulosa com uma taxa de aleatoriedade $\alpha \in [0,1]$. Uma seqüência de tarefas (solução) deve ser

construída inserindo uma tarefa por vez numa seqüência parcial s. A seqüência s inicialmente é vazia ($s = \varnothing$).

Como estratégia gulosa é usada uma regra de despacho similar à regra proposta em [16] para minimizar o atraso total das tarefas no problema de escalonamento de tarefas em uma máquina com tempos de preparação (*setup-times*). A regra utilizada consiste em ordenar as tarefas de acordo com a seguinte função de custo:

$$Custo(i) = w_i \times \{d_i - (C_k + p_i)\} \times p_i,$$

onde, d_i, p_i e w_i são, respectivamente, a data de entrega, o tempo de processamento e a prioridade da tarefa i candidata a ser inserida na seqüência parcial s. C_k é o tempo de finalização da última tarefa k inserida na seqüência parcial. Se a tarefa i fosse inserida em s, tem-se $s = \{1,...,k\}+\{i\} = \{1,...,k, i\}$.

Na construção de uma solução, inicialmente as tarefas são ordenadas em uma lista de candidatos (LC) usando a função $Custo(i)$. Note que, as tarefas com menores valores da função *Custo* (principalmente as tarefas com custo negativo) terão prioridade para serem processadas primeiro. A partir de uma sub-lista de tarefas LCR (formada pelas $|LCR|$ primeiras tarefas de LC), seleciona-se aleatoriamente uma tarefa i para ser inserida na seqüência parcial s. Remove-se a tarefa i de LC, reordenam-se as tarefas (utilizando a função *Custo*) e o processo é repetido escolhendo uma nova tarefa i. A construção finaliza quando a seqüência s fique completa, ou seja, quando ela contenha todas as n tarefas. O tamanho da sub-lista LCR depende da taxa de aleatoriedade α utilizada: $|LCR| = \max(1, \alpha \times |LC|)$. Note que, se a taxa $\alpha = 0$, a escolha da tarefa i é puramente gulosa, ou seja, escolhe-se sempre a primeira tarefa da lista LC (a tarefa com o menor valor de *Custo*). Se $\alpha = 1$, a escolha da tarefa i é totalmente aleatória.

Na ordenação das tarefas na lista LC, se existirem tarefas com o mesmo valor de *Custo* (existência de empate), o critério de desempate será dada pelo tempo de processamento, ou seja, tarefas com menor tempo de processamento terão preferência.

2.2 Busca Local

A Busca Local é um procedimento iterativo que consiste em melhorar uma solução s_1 procurando novas soluções vizinhas dela. Estas soluções vizinhas são obtidas realizando algumas alterações (*movimentos*) na estrutura da solução atual s_1. Escolhe-se um vizinho s dentre todos os vizinhos de s_1. Se o vizinho escolhido s é melhor que s_1, a busca continua a partir de s (ou seja, $s_1 \leftarrow s$). O procedimento finaliza quando não seja possível melhorar a solução atual s_1, ou seja, quando s_1 é um ótimo local.

Neste trabalho foram utilizados três tipos de movimentos para a obtenção de soluções vizinhas:

- *Troca*: uma solução vizinha de s_1 é gerada fazendo a troca de duas tarefas i e j da seqüência, $1 \le i,j \le n$, $i \ne j$. O número de vizinhos gerados com este movimento é $n(n-1)/2$.
- *Inserção à esquerda*: um vizinho de s_1 é gerado inserindo uma tarefa que está na posição i da seqüência em outra posição j, tal que $j<i$, $2 \le i \le n$, $1 \le j \le n-1$. Usando este movimento também são gerados $n(n-1)/2$ vizinhos.

- *Inserção à direita*: um vizinho de s_1 é gerado inserindo uma tarefa que está na posição i da seqüência, em outra posição j, tal que $j>i$, $1 \leq i \leq n-1$, $2 \leq j \leq n$. Usando este movimento é possível gerar $n(n-1)/2$ vizinhos.

Exemplo: Seja uma solução (seqüência) s_1 = {1, 2, 3, 4}. Os vizinhos de s_1 gerados usando *trocas* são: {**2**, **1**, 3, 4,}, {**3**, 2, **1**, 4}, {**4**, 2, 3, **1**}, {1, **3**, **2**, 4}, {1, **4**, 3, **2**} e {1, 2, **4**, **3**}. Os vizinhos de s_1 gerados usando *inserções à esquerda* são: {**2**, 1, 3, 4,}, {**3**, 1, 2, 4}, {1, **3**, 2, 4}, {**4**, 1, 2, 3}, {1, **4**, 2, 3} e {1, 2, **4**, 3}. Os vizinhos de s_1 gerados usando *inserções à direita* são: {2, **1**, 3, 4,}, {2, 3, **1**, 4}, {2, 3, 4, **1**}, {1, 3, **2**, 4}, {1, 3, 4, **2**} e {1, 2, 4, **3**}.

Na Busca Local do algoritmo GRASP-PR, a escolha da próxima solução vizinha (s) a ser explorada é baseada no "*melhor vizinho*", ou seja, dentre todos os vizinhos gerados, fazendo trocas e inserções (à esquerda e à direita), o melhor vizinho s é escolhido para ser a solução atual ($s_1 \leftarrow s$). O procedimento de Busca Local finaliza quando a melhor solução vizinha não seja melhor que a solução atual s_1.

2.3 Intensificação com Path Relinking

Path Relinking é uma técnica bastante usada com outras metaheurísticas tais como busca tabu, *scatter search* [14] e GRASP ([17], [24]). A técnica *Path Relinking* possui como entrada duas soluções s_1 e s_2 denominadas de origem e guia, respectivamente. A idéia é construir um caminho de s_1 para s_2 com o objetivo de encontrar novas soluções com características de s_1 e s_2. Este caminho é obtido transformando gradativamente s_1 em s_2.

Neste trabalho uma seqüência de tarefas s_1 é transformada na seqüência s_2 realizando troca de tarefas. Inicialmente, são identificadas as tarefas que estão nas mesmas posições s_1 e s_2. Essas tarefas não serão movimentadas. As tarefas de s_1, com posições diferentes em relação a s_2, são movimentadas. Desta maneira, gera-se um conjunto de seqüências vizinhas. Para continuar a trajetória, é escolhida a seqüência com menor atraso total ponderado. A cada passo são obtidas soluções mais diferentes da seqüência "origem" s_1 e mais similares à seqüência "guia" s_2.

Exemplo: Sejam as seqüências (soluções) origem e guia: s_1 = {**1**, 2, 3, 4} e guia s_2 = {**1**, 4, 2, 3}. Note que s_1 e s_2 possuem uma tarefa na mesma posição. Em s_1, os pares de tarefas a serem trocadas são 2-3, 2-4 e 3-4. São obtidos os seguintes vizinhos: {1, 3, 2, 4}, {1, 4, 3, 2} e {1, 2, 4, 3}. Escolhe-se o melhor vizinho (ou seja, a seqüência com menor atraso total ponderado). Suponha que {**1**, 2, 4, **3**} seja escolhido. Note que esta seqüência, possui duas tarefas em comum (na mesma posição) com s_2. A partir de {**1**, 2, 4, **3**} pode ser feita somente uma troca 2-4. Fazendo esta troca obtém-se s_2 = {**1**, **4**, **2**, **3**}.

No *Path Relinkin*g, a seqüência s_1 (origem) será a seqüência correspondente à solução obtida após a Busca Local e a seqüência guia s_2 será a melhor obtida até a iteração atual. Na primeira iteração da GRASP não será executada a intensificação, pois não haverá uma solução guia para ser utilizada.

3 Testes Computacionais

A metaheurística GRASP-PR proposto neste trabalho foi programada na linguagem Java e os testes computacionais foram executados num computador Pentium IV, 3,0 Ghz. A metaheurística foi testada em 375 problemas disponíveis na biblioteca virtual OR-*Library* [3]. Existem três tamanhos de problemas, na qual o número de tarefas são $n = 40$, $n = 50$ e $n = 100$. Para cada n existem 125 problemas. Dos 125 problemas com $n = 40$ e $n = 50$, na OR-Library são disponibilizados, 124 e 115 soluções ótimas, respectivamente. Para os problemas com $n = 100$, são disponibilizadas as melhores soluções obtidas pelo algoritmo *Iterated Dynasearch* proposto em [6].

Os dois únicos parâmetros da metaheurística GRASP são: número máximo de iterações (I_{max}) e a taxa de aleatoriedade usada na fase construtiva (α). Foram testados diferentes valores para estes parâmetros. Os valores de parâmetros que permitiram a estabilidade da metaheurística e gerando bons resultados foram: $I_{max} = 200$ e $\alpha = 0{,}3$.

Tabela 1. Comparação das soluções da metaheurística GRASP-PR com as soluções da OR-*Library*.

Problemas	OR-*Library* T_w	GRASP-PR T_w	GRASP-PR *Erro%*
$n = 40$: Problema 3	537	573	6,70
$n = 50$: Problema 12	36.378	36.459	0,22
$n = 50$: Problema 63	30.729	30.737	0,03
$n = 50$: Problema 85	3.780	3.796	0,42
$n = 50$: Problema 107	1.717	1.798	4,72
$n = 50$: Problema 109	6.185	6.195	0,16
$n = 100$: Problema 15	172.995	173.020	0,01
$n = 100$: Problema 20	406.094	406.145	0,01
$n = 100$: Problema 39	151.701	153.082	0,91
$n = 100$: Problema 56	9.046	9.478	4,78
$n = 100$: Problema 63	96.563	96.652	0,09
$n = 100$: Problema 86	66.850	67.909	1,58
$n = 100$: Problema 88	55.544	55.560	0,03
Média	79.855,3	80.108,0	1,51

Para os problemas com $n = 40$, 50 e 100 tarefas, o algoritmo GRASP-PR obteve, respectivamente, 124, 120 e 118 soluções iguais às soluções disponíveis na OR-*Library*. Isto é, o algoritmo GRASP-PR obteve em total 362 soluções das 375 melhores soluções disponíveis na OR-*Library*. Vale mencionar que a metaheurística GRASP-PR foi levemente melhor que a versão básica da metaheurística, composta somente pelas fases de construção e busca local (denotada por GRASP-B). A versão básica GRASP-B obteve 122, 116 e 105 soluções da OR-*Library* para os problemas com $n = 40$, 50 e 100, respectivamente. Ou seja, para o total de 375 problemas, o algoritmo GRASP-B obteve 343 soluções disponíveis na OR-*Library*.

Nos 13 problemas onde o algoritmo GRASP+PR não obteve a mesma solução da OR-*Library*, o erro relativo porcentual médio foi de apenas 1,5%. Na Tabela 1

apresentam-se as comparações dos valores do atraso total ponderado (T_w) e o erro relativo porcentual produzido, para cada um dos 13 problemas. Observa-se que das 13 soluções mostradas na Tabela, o erro porcentual é menor que 1% para 9 soluções.

O tempo computacional gasto pelo algoritmo GRASP-PR foi relativamente baixo. Para problemas com n = 40, 50 e 100 tarefas, o tempo médio de execução do algoritmo foi de 4, 14 e 174 segundos, respectivamente.

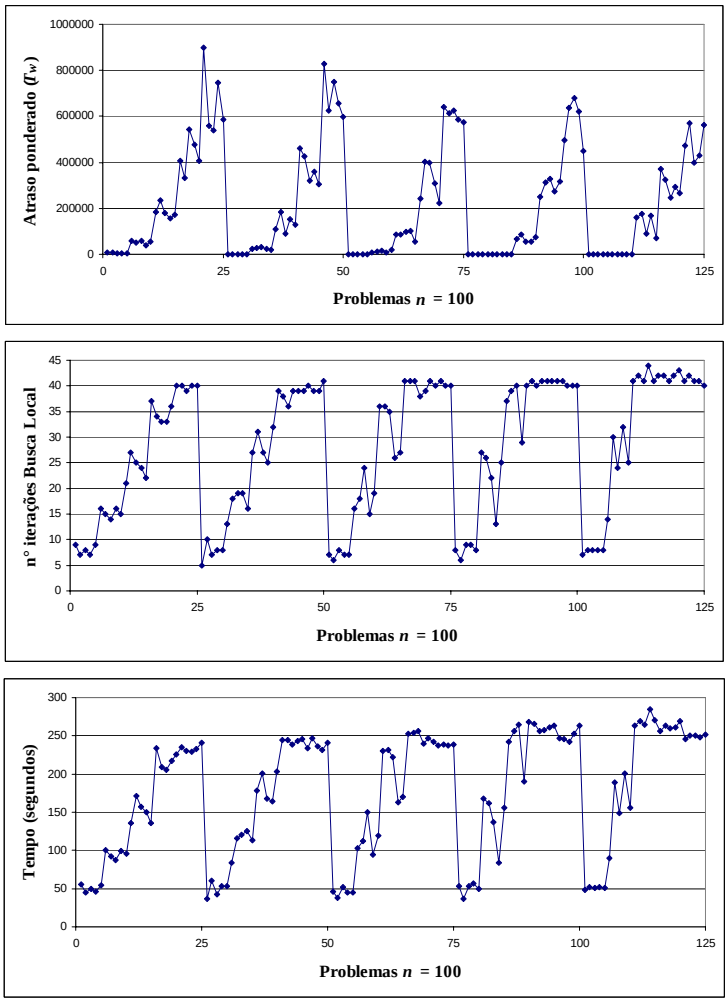

Fig. 3. Variação da função objetivo (T_w), número de iterações da busca local e tempo computacional do algoritmo.

Foi observado que, os problemas testes da OR-*Library* estão agrupados em grupos de 25 problemas (1 a 25, 26 a 50,...., 101 a 125). Em cada grupo, os problemas são ordenados em ordem crescente do valor do atraso total ponderado T_w. O tempo de execução do algoritmo varia de acordo com o valor de T_w. Esta variação do tempo computacional deve-se ao seguinte fato: o número de iterações gastos para melhorar uma solução, na fase da busca local, depende do valor de T_w. Se o atraso total

ponderado é baixo, uma solução será melhorada rapidamente gastando poucas iterações, caso contrário serão necessárias varias iterações. Na Figura 3 ilustram-se a variação do valor de T_w, o número de iterações da busca local e tempo computacional do algoritmo GRASP-PR, para os 125 problemas com $n = 100$ tarefas.

O desempenho do algoritmo GRASP-PR também é comparado com um algoritmo Busca Tabu proposto recentemente em por Bilge, Kurtulan e Kiraç (2007) [4] que mostrou ser bastante eficiente na resolução do problema SMTWT. Na Tabela 2 compara-se o número das melhores soluções encontradas pelos algoritmos GRASP-PR e Busca Tabu de Bilge, Kurtulan e Kiraç. Note que a Busca Tabu obteve em total 357 soluções, já o algoritmo GRASP-PR encontrou 362 soluções. Os tempos médios de CPU gastos pela melhor versão do algoritmo Busca Tabu foram maiores em comparação aos tempos do algoritmo GRASP-PR, devido a que a Busca Tabu foi executada em um computador diferente (Pentium IV, 1.6 Ghz). Para problemas com $n = 40$, 50 e 100 tarefas, o tempo médio de execução da melhor versão do algoritmo Busca Tabu foi de 28,46, 170,71 e 283,46 segundos, respectivamente.

Tabela 2. Comparação do algoritmo GRASP-PR com o algoritmo Busca Tabu de Bilge, Kurtulan e Kiraç (2007).

Algoritmos	Número de melhores soluções encontradas			Total
	$n = 40$	$n = 50$	$n = 100$	
GRASP-PR	124	120	118	362
Busca Tabu	125	124	108	357

4 Conclusões

Neste trabalho foi proposto um algoritmo GRASP para resolver o problema de escalonamento de tarefas em uma máquina, minimizando o atraso total ponderado. O algoritmo possui uma fase de intensificação baseada na técnica de *Path Relinking* que mostrou ser eficiente. Utilizando o algoritmo GRASP com *Path Relinking* (GRASP-PR) foram encontrando 362 soluções das 375 melhores soluções disponíveis na OR-*Library*. Nas 13 soluções onde não foi encontrada a melhor solução disponível, o erro relativo porcentual médio foi de apenas 1,5%. O algoritmo GRASP-PR foi bastante competitivo quando comparando com um algoritmo Busca Tabu recentemente proposto na literatura. O algoritmo Busca Tabu obteve 357 melhores soluções, cinco a menos que o algoritmo GRASP-PR. O objetivo deste trabalho foi demonstrar a eficiência da metaheurística GRASP na solução do problema SMTWT, proporcionando assim uma ferramenta de auxilio a decisão na área de programação e planejamento da produção.

Agredecimentos. Este trabalho foi financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq (processo: 480096/2007-4) e pela Fundação de Amparo à Pesquisa do Estado de Minas Gerais – FAPEMIG (processo: APQ3768-6.01/07).

Referências

1. Abdul-Razaq, T.S., Potts, C.N., Van Wassenhove, L.N.: A survey of algorithms for the single machine total weighted tardiness scheduling problem. Discrete Applied Mathematics 26, 235–253 (1990).
2. Armentano, V. A., Araújo, O. C. B.: Grasp with memory-based mechanisms for minimizing total tardiness in single machine scheduling with setup times, Journal of Heuristics, 12:427–446 (2006).
3. Beasley, J.E. OR-Library. Disponível em:<http://mscmga.ms.ic.ac.uk/jeb/orlib/wtinfo.html> (2001).
4. Bilge, U., Kurtulan, M., Kıraç, F.: A tabu search algorithm for the single machine total weighted tardiness problem, European Journal of Operational Research, 176, 1423–1435 (2007).
5. Cheng, T.C.E., Ng, C.T., Yuan, J.J., Liu, Z.H.: Single machine scheduling to minimize total weighted tardiness. European Journal of Operational Research 165, 423–443 (2005).
6. Congram, R.K., Potts, C.N., Van de Velde, S.L.: An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. Informs Journal on Computing 14 (1), 52–67 (2002).
7. Crauwels, H.A.J., Potts, C.N., Van Wassenhove, L.N.: Local search heuristics for single machine total weighted tardiness scheduling problem. Informs Journal on Computing, 10, 341–350 (1998).
8. Den Besten, M. Stützle,T. & Dorigo M.: An Ant Colony Optimization Application to the Single Machine Total Weighted Tardiness Problem, In: VI 6th International Conference - Parallel Problem Solving from Nature – PPSN (2000).
9. Du, J., Leung, J.Y.T.: Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research, 15, 483–495 (1990).
10. Feo, T., Resende, M.G.C.: Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization 2, 1–27 (1995).
11. Ferrolho, A., Crisostomo, M.: Single Machine Total Weighted Tardiness Problem with Genetic Algorithms, IEEE/ACS International Conference on Computer Systems and Applications, 1-8 (2007).
12. Festa, P., Resende, M.G.C.: An Annotated Bibliography of GRASP. European Journal of Operational Research, submitted (2004).
13. Fisher, M. L.: A dual algorithm for the one machine scheduling problem. Mathematical Programming, 11, 229–252 (1976).
14. Glover, F., Laguna, M., Martí, R.: (2000). Fundamentals of scatter search and path relinking, Control and Cybernetics, 39, 653–684.
15. Glover, F.: Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges. Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, Kluwer, 1-75 (1996).
16. Gupta, S.R., Smith, J.S.: Algorithms for single machine total tardiness scheduling with sequence dependent setups. European Journal of Operational Research, 175: 722–739 (2006).
17. Laguna M., Martí, R.: GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing, 11, 44-52 (1999).
18. Lawler, E.L.: A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics 1, 331–342 (1977).
19. Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P.: Complexity of machine scheduling problems. Annals of Discrete Mathematics 1, 343–362 (1977).

20. Matsuo, H., Suh, C.J., Sullivan, R.S.: A controlled search simulated annealing method for the single machine weighted tardiness problem. Annals of Operations Research 21, 85–108 (1989).
21. Panwalkar, S.S., Smith, M.L., Koulamas, C.P.: A heuristic for the single machine tardiness problem. European Journal of Operations Research 70, 304–310 (1993).
22. Potts, C.N., Van Wassenhove, L.N.: Single machine tardiness sequencing heuristics. IIE Transactions 23, 346–354 (1991).
23. Rachamadugu, R.V., Morton, T.E.: Myopic Heuristics for the Single Machine Weighted Tardiness Problem, Paper 30-82-83, GSIA, Carnegie Mellon University, Pittsburgh, PA (1982).
24. Resende, M. G. C., Ribeiro, C.C.: GRASP with path-relinking: Recent advances and applications, in: T. Ibaraki, K. Nonobe, M. Yagiura (Eds.), Metaheuristics: Progress as Real Problem Solvers, Springer, p. 29–63 (2005).