Proceeding Series of the Brazilian Society of Applied and Computational Mathematics, Vol. 3, N. 1, 2015.
Trabalho apresentado no XXXV CNMAC, Natal-RN, 2014.

Algoritmos *Branch* e *Bound* para o problema de sequenciamento em uma única máquina

Carlos E. Ferreira
Instituto de Matemática e Estatística, IME, USP
05508-090, São Paulo, SP
E-mail: cef@ime.usp.br

Wellington D. Previero
UTFPR - Departamento de Matemática / Instituto de Matemática e Estatística, IME, USP
86036-370, Londrina, PR / 05508-090, São Paulo, SP
E-mail: previero@utfpr.edu.br

Resumo: *Neste trabalho apresentamos o problema de minimização do makespan para o sequenciamento de tarefas em uma única máquina. Para este problema, cada tarefa está associada a um tempo de execução, um instante de disponibilidade de entrada no sistema e o tempo em que a tarefa fica no sistema após o seu processamento. Para a resolução do problema foram utilizados dois algoritmos baseados no método Branch e Bound e os resultados foram comparados. O primeiro algoritmo utiliza as estratégias de Carlier [2] e o segundo as de Grabowski, Nowicki e Zdrzalka [5]. O limitante inferior para o Branch e Bound é obtido através do Algoritmo Preemptivo de Jackson.*

Palavras-chave: *Sequenciamento em uma única máquina, escalonamento de tarefas*

1 Introdução

O problema de sequenciamento em uma máquina (PSUM) consiste em ordenar um conjunto de tarefas para serem processadas em uma única máquina de modo a minimizar o tempo de conclusão da última tarefa a sair do sistema. Neste problema, cada tarefa está associada a um tempo de execução na máquina, um instante de disponibilidade de entrada no sistema e o tempo em que a tarefa fica no sistema após o seu processamento. Uma tarefa está finalizada assim que sair do sistema. Após o seu início, uma tarefa é processada até o seu término, sem interrupção.

O problema definido acima tem várias aplicações em sistemas de produção. Os resultados obtidos para o PSUM não só fornecem o conhecimento para o problema em si, como também podem proporcionar heurísticas a ambientes mais complexos, como o *Job Shop* ou o problema de sequenciamento de máquinas em paralelo. Para o *Job Shop*, podemos usar o PSUM para obter limitantes. Neste caso, uma máquina é escolhida e através da relaxação das restrições de outras máquinas obtemos limitantes para o problema. Isso nos ajuda a resolver uma das principais dificuldades do método *Branch* e *Bound* que é a ausência de bons limitantes inferiores para podar a árvore de enumeração tão cedo quanto o possível.

O PSUM é um problema Otimização Combinatória de difícil resolução. Garey e Johnson [4] e Lenstra, Kan e Brucker [9] mostraram que este problema é fortemente NP-difícil.

Carlier [2] desenvolveu um algoritmo *Branch* e *Bound* para resolver o problema de sequenciamento em uma máquina. As soluções factíveis são obtidas através do algoritmo de Schrage [12]. São apresentadas duas propostas de limitantes inferiores e as estratégias de ramificação são definidas a partir de escolha de uma tarefa específica e de um subconjunto de tarefas. Carlier e Pinson [3] desenvolveram um algoritmo *Branch* e *Bound* para o problema *Job Shop* baseado

DOI: 10.5540/03.2015.003.01.0431

na resolução do PSUM. O algoritmo foi o primeiro a resolver um importante problema com 10 máquinas e 10 *jobs* proposto por Muth e Thompson [11].

Em Grabowski, Nowicki e Zdrzalka [5] é proposto um novo algoritmo *Branch* e *Bound* para resolver o PSUM. A abordagem utilizada é baseada em propriedades das tarefas pertencentes a um subconjunto de tarefas denominado bloco crítico. Estas propriedades definem a maneira como a alteração da ordem de uma tarefa em um sequenciamento pode influenciar na solução corrente. As propriedades definidas por Grabowski, Nowicki e Zdrzalka [5] também são utilizadas na resolução do problema de *Job Shop* [1], [6] e [3].

Liaw [10] apresenta um algoritmo *Branch* e *Bound* para o PSUM no qual em cada tarefa é atribuída uma penalização caso seja finalizada antes ou após um tempo limite. O limitante inferior de cada subproblema é obtido através da Relaxação Lagrangeana. O limite superior é obtido usando uma heurística de duas fases que combina uma regra de envio de prioridades com um processo de melhoria local.

O objetivo deste trabalho consiste em utilizar as estratégias de Carlier [2] e Grabowski, Nowicki e Zdrzalka [5] na resolução do PSUM e comparar os resultados obtidos.

2 Descrição do Problema

É dado um conjunto de n tarefas independentes (*jobs*) indexadas $I = \{1, \ldots, n\}$ que devem ser processadas em uma única máquina. Para cada tarefa $i \in I$, sejam $r_i \geq 0$ o instante de disponibilidade da tarefa para a entrada no sistema, $p_i > 0$ o tempo de processamento na máquina e $q_i \geq 0$ o tempo em que a tarefa fica no sistema após o seu processamento. Desejamos determinar para cada tarefa $i \in I$ o instante de início t_i de seu processamento na máquina. Assim, $t_i \geq r_i$, para todo $i \in I$. Temos conhecimento de toda a informação sobre a tarefa antes da fase de sequenciamento (o problema é *offline*). O problema consiste em definir a ordem entre as tarefas que devem ser processadas na máquina de modo a minimizar o *makespan*, isto é, minimizar o tempo de conclusão da última tarefa a sair do sistema. Sejam $\pi = [\pi(1), \ldots, \pi(n)]$ o vetor permutação de I e Π o conjunto de todas as permutações. Sendo $C_i = t_i + p_i$ o tempo de conclusão da tarefa i, o problema consiste em encontrar a permutação $\pi \in \Pi$ que minimiza $C_{Max}(\pi) = Max_{1 \leq i \leq n}(C_i + q_i)$, para todo $\pi \in \Pi$. Este problema é denotado por $1|r_i, q_i|C_{Max}$.

Dado um vetor permutação π, podemos associar a ordem de sequenciamento das tarefas de π a um grafo $G = (V, A)$. O conjunto V de vértices representa as tarefas $i \in I$, além das tarefas fictícias O e F, que descrevem o início e o término do processamento, respectivamente. O conjunto A de arcos é composto por três conjuntos: $A = U_1 \cup U_2 \cup U_3$, tal que $U_1 = \{(O, i),\ \ i \in I\}$, $U_2 = \{(i, F),\ \ i \in I\}$ e $U_3 = \{(i, j),$ se i precede j no sequenciamento em $\Pi\}$. A cada arco $(O, i) \in U_1$ é atribuído o valor r_i e para cada arco $(i, F) \in U_2$ é atribuído o valor $q_i + p_i$. Além disso, para cada arco $(i, j) \in U_3$ é atribuído o valor p_i.

O instante de processamento t da tarefa corresponde ao valor do maior caminho ponderado entre o nó O e o nó i. Logo t_F corresponde ao valor do *makespan* e

$$C_{Max}(\pi) = r_{\pi(i_1)} + \sum_{i=\pi(i_1)}^{\pi(i_2)} p_i + q_{\pi(i_2)}$$

para algum par (i_1, i_2) com $1 \leq i_1 \leq i_2 \leq n$ e $\pi(i_1)$, $\pi(i_2) \in I$. A sequência $[\pi(i_1), \pi(i_1 + 1), \ldots, \pi(i_2)]$ de tarefas sucessivas de π é denominada de bloco crítico de π e será denotado por B_π. As tarefas $\pi(i_1)$ e $\pi(i_2)$ representam a primeira e a última tarefa do bloco, respectivamente.

3 Método *Branch* e *Bound*

Nesta seção apresentaremos dois procedimentos para a resolução do PSUM através do método *Branch* e *Bound*. No primeiro método é utilizada a proposta de Carlier [2] (Algoritmo 1) e no

DOI: 10.5540/03.2015.003.01.0431

segundo a de Grabowski, Nowicki e Zdrzalka [2] (Algoritmo 2). Para cada método, definiremos os procedimentos para o cálculo dos limitantes superior e inferior e a estratégia de ramificação.

3.1 Algoritmo 1

O limitante superior para o problema é determinado através do Algoritmo de Schrage. O algoritmo em consiste em dividir, em cada iteração, o conjunto de tarefas I em dois conjuntos: o conjunto de tarefas sequenciadas (S) e o conjunto de tarefas não sequenciadas (N). Inicialmente temos que $S = \emptyset$ e $N = I$ e t é inicializado com $Min_{i \in N} r_i$. No tempo t, dentre as tarefas $i \in N$ tais que $r_i \leq t$, selecionamos a que possui o maior valor de q_i. A tarefa i selecionada é removida do conjunto N e adicionada ao conjunto S. O valor de t é atualizado para $t = Max(t_i + p_i, Min_{i \in N} r_i)$. O algoritmo é finalizado quando todas as tarefas do conjunto N são sequenciadas, isto é, $N = \emptyset$. Na próxima proposição mostramos como obter o limitante inferior para o valor do *makespan*.

Proposição 1. Para todo $I_t \subseteq I$, temos que

$$h(I_t) = Min_{i \in I_t} r_i + \sum_{i \in I_t} p_i + Min_{i \in I_t} q_i$$

é um limitante inferior para o valor ótimo do *makespan*.

Limitantes inferiores mais justos que o apresentado na Proposição 1 podem ser obtidos, por exemplo, através do Algoritmo Preemptivo de Jackson [7]. Neste algoritmo um escalonamento das tarefas é obtido permitindo preempção, ou seja, uma tarefa pode ser interrompida em qualquer instante. A ideia é então, seguir a mesma ordenação proposta no algoritmo de Scharage (maior valor de q_i entre as tarefas disponíveis para execução) interrompendo a execução de uma tarefa sempre que outra tarefa j tiver $q_j > q_i$. Desta forma, minimizamos o tempo ocioso da máquina. Obviamente isso é uma relaxação do nosso problema (em que a preempção não é permitida), e portanto o valor do *makespan* obtido é um limitante inferior para o *makespan* ótimo quando preempção não é permitida.

A complexidade do Algoritmo de Schrage e do Algoritmo Preemptivo de Jackson é $O(n \log n)$ [2].

Proposição 2. O *makespan* do Algoritmo Preemptivo de Jackson é um limitante inferior para o *makespan* ótimo do problema de sequenciamento de uma única máquina. Além disso, o seu valor é igual a $Max_{I_t \subset I} h(I_t)$.

O próximo teorema implica que se o sequenciamento obtido pelo algoritmo de Scharage não for ótimo, então está "próximo"do ótimo, e para melhorá-lo, uma tarefa específica $\pi(k)$ tem que ser escolhida.

Proposição 3. Sejam π e L o sequenciamento e o makespan obtido pelo algoritmo de Schrage para as tarefas do conjunto I, respectivamente. Se o sequenciamento π é ótimo, então existe um conjunto J tal que $h(J) = L$. Caso contrário, existe uma tarefa crítica $\pi(k)$ e um conjunto crítico J tal que $h(J) = Min_{\pi(j) \in J} r_{\pi(j)} + \sum_{\pi(j) \in J} p_{\pi(j)} + Min_{\pi(j) \in J} q_{\pi(j)} > L - p_{\pi(k)}$.

A tarefa $\pi(k)$ e o conjunto J são obtidos da seguinte forma. Sejam π e L o sequenciamento e o *makespan* obtido pelo algoritmo de Schrage, respectivamente. Suponha que B_π seja o bloco crítico definido por π com um número máximo de tarefas. Se $h(B_\pi) = L$, então o sequenciamento é ótimo. Logo para todos $\pi(i), \pi(j) \in B_\pi$, com $i_1 \leq i < j \leq i_2$, temos que $q_{\pi(i)} \geq q_{\pi(j)}$. Se π não é ótimo, então existe $i < i_2$ tal que $q_{\pi(i)} < q_{\pi(i_2)}$. A tarefa $\pi(k)$ é escolhida como sendo a última tarefa de B_π tal que $q_{\pi(k)} < q_{\pi(i_2)}$ e o conjunto J é definido pro $J = \{\pi(k+1), \ldots, \pi(i_2)\}$. O corolário seguinte define a estratégia de ramificação utilizado pelo Algoritmo 1.

Corolário 1. Se $h(J) > L - p_{\pi(k)}$, então a distância entre o *makespan* ótimo e o *makespan* do sequenciamento do Algoritmo de Schrage é menor que $p_{\pi(k)}$. Além disso, no sequenciamento ótimo, ou $\pi(k)$ será processado antes que todas as tarefas de J ou $\pi(k)$ será processado após todas as tarefas de J.

A cada nó do algoritmo *Branch* e *Bound* é associado um PSUM, que fornece um limitante inferior f_{Min}. O Algoritmo Preemptivo de Jackson é utilizado para o cálculo do limitante inferior. O limitante superior f_{Max} é o atualizado por $f_{Max} = Min\{f_{Max}, L\}$, onde L é o *makespan* do sequenciamento produzido pelo algoritmo de Schrage no nó atual. O nó com o menor limitante inferior na árvore de busca é selecionado e aplica-se o algoritmo de Schrage para gerar o sequenciamento das tarefas. Se não for possível determinar uma tarefa crítica $\pi(k)$ para o sequenciamento obtido, então o sequenciamento é ótimo. Caso contrário, temos dois novos problemas. O primeiro problema em que a tarefa $\pi(k)$ é sequenciada antes de todas as tarefas do conjunto crítico J e o segundo problema em que a tarefa $\pi(k)$ é forçada a ser sequenciada após todas as tarefas do conjunto crítico J. Se $\pi(k)$ é processado antes de todas as tarefas de J, o valor de $q_{\pi(k)}$ pode ser atualizado para $q_{\pi(k)} = Max\left(q_{\pi(k)}, \sum_{\pi(j)\in J} p_{\pi(j)} + Min_{\pi(j)\in J} q_{\pi(j)}\right)$. Para que a tarefa $\pi(k)$ seja processada após todas as tarefas de J, podemos fazer que $r_{\pi(k)} = Max\left(r_{\pi(k)}, Min_{\pi(j)\in J} r_{\pi(j)} + \sum_{\pi(j)\in J} p_{\pi(j)}\right)$. Estas alterações ficam fixas para o nó e todos os seus descendentes. Os novos nós serão acrescentados na árvore somente quando o limitante inferior for menor que o limitante superior.

3.2 Algoritmo 2

Se π representa uma permutação das tarefas de I, então o valor $C_{Max}(\pi)$ define um limitante superior para o *makespan* do sequenciamento ótimo. O limitante inferior utilizado pelo Algoritmo 2 é o descrito pela Proposição 2. O próximo teorema define a estratégia de ramificação do método *Branch* e *Bound*.

Proposição 4. Sejam π uma permutação de tarefas do conjunto I e B_π o conjunto de tarefas do bloco crítico da permutação π. Se β é qualquer permutação de I tal que $C_{Max}(\beta) < C_{Max}(\pi)$, então existe uma tarefa $\pi(k) \in B_\pi - \{\pi(i_1)\}$ (ou $\pi(k) \in B_\pi - \{\pi(i_2)\}$) e na permutação β a tarefa $\pi(k)$ precede (ou sucede) todas as tarefas do conjunto $B_\pi - \{\pi(k)\}$.

Através da Proposição 4 podemos construir, a partir de um sequenciamento π, um possível sequenciamento de melhor *makespan* analisando dois subproblemas para as tarefas de seu bloco crítico: no primeiro subproblema a tarefa é executada antes (posição i_1) de todas as tarefas do bloco crítico, e no segundo depois de todas as outras tarefas do bloco (posição i_2).

As tarefas a serem deslocadas para a posição i_1 serão tomadas do conjunto $T_b = \{\pi(j) \in B_\pi; \Delta_b(\pi(j)) < 0\}$, tal que $\Delta_b(\pi(j)) = p_{\pi(j)} - p_{\pi(i_1)}$; as tarefas deslocadas para a posição i_2 serão tomadas do conjunto $T_a = \{\pi(j) \in B_\pi; \Delta_a(\pi(j)) < 0\}$, com $\Delta_a(\pi(j)) = q_{\pi(j)} - q_{\pi(i_2)}$.

Seja β o sequenciamento obtido a partir de π através do deslocamento da tarefa $\pi(k) \in T_b$ ($\pi(k) \in T_a$). Assim, todos os descendentes de β devem ter a propriedade de que a tarefa $\pi(k)$ seja processada antes (após) de todas as tarefas do conjunto $T_b^{'} = T_b \cup \{\pi(i_1)\}$ ($T_a^{'} = T_a \cup \{\pi(i_2)\}$). Logo, cada nó da árvore está associado a um conjunto de restrições de prioridades. Seja R_π o conjunto de restrições de prioridades associado a π. Para que as restrições de R_π sejam respeitadas para os novos descendentes de π, o conjunto T_b é redefinido para $T_b = \{\pi(j) \in B_\pi; \Delta_b(\pi(j)) < 0$ e $\pi(j)$ não é executada após a tarefa $\pi(k)$ em $R_\pi, k = i_1, \ldots, j-1\}$. De forma análoga, o conjunto T_a é redefinido por $T_a = \{\pi(j) \in B_\pi; \Delta_a(\pi(j)) < 0$ e $\pi(j)$ não é executada antes da tarefa $\pi(k)$ em $R_\pi, k = j+1, \ldots, k\}$.

Quando a tarefa $\pi(k)$ é deslocada para a posição i_1 os valores de r_i e q_i são atualizados para $r_{\pi(j)} = Max\left(r_{\pi(j)}, r_{\pi(k)} + p_{\pi(k)}\right)$, para todo $\pi(j) \in T_b^{'}$, $\pi(j) \neq \pi(k)$, e $q_{\pi(k)} = Max(q_{\pi(k)}, \sum_{\pi(j)\in T_b^{'}} p_{\pi(j)} - p_{\pi(k)} + Min_{\pi(j)\in T_b^{'}\setminus\{\pi(k)\}} q_{\pi(j)})$.

Agora, uma vez que a tarefa $\pi(k)$ é deslocada para a posição i_2, os valores de r_i e q_i são fixados em $r_{\pi(k)} = Max\left(r_{\pi(k)}, \sum_{\pi(j)\in T'_a} p_{\pi(j)} - p_{\pi(k)} + Min_{\pi(j)\in T'_a\setminus\{\pi(k)\}} r_{\pi(j)}\right)$ e $q_{\pi(j)} = Max\left(q_{\pi(j)}, p_{\pi(k)} + q_{\pi(k)}\right)$ para todo $\pi(j) \in T'_a$ e $\pi(j) \neq \pi(k)$.

Para iniciar o método *Branch* e *Bound*, uma permutação inicial π é obtida através do Algoritmo de Schrage. Os nós descendentes da permutação π são obtidos através dos deslocamentos das tarefas nos conjuntos T_b e T_a e adicionados a árvore de pesquisa. Cada nó descendente de π representa uma nova permutação de tarefas de I. O nó com o menor limitante inferior na árvore de busca é selecionado e novos possíveis descendentes são gerados. Grabowski, Nowicki e Zdrzalka [5] utilizam a estratégia de *Backtracking* para a resolução do PSUM.

4 Experimentos Computacionais

O algoritmo *Branch* e *Bound* foi implementado em linguagem Java versão 1.7 utilizando Eclipse IDE versão Juno Service Release 2. Os testes foram realizados em um computador Acer Intel Core i5/2.53 GHz e 4 GB de memória ram. Os problemas testes em Carlier [2] e Grabowski, Nowicki e Zdrzalka [5] foram gerados através da metodologia descrita por Lentra [8]. Desta forma utilizamos o mesmo procedimento para gerar os problemas testes. Definido o valor de n para o número de tarefas do PSUM, os valores r_i, q_i e p_i foram gerados a partir de uma distribuição uniforme entre 1 e r_{Max}, 1 e q_{Max} e 1 e d_{Max}, respectivamente. Os valores r_{Max}, q_{Max} e d_{Max} foram definidos por $r_{Max} = 50R$, $q_{Max} = 50Q$ e $d_{Max} = 50$, com $R, Q = \{0,5; 2; 0,5n, 2n\}$. Para cada valor de $n = \{20, 40, 80, 150, 200\}$, 80 testes foram gerados, sendo 5 testes para cada combinação dos parâmetros R e Q. O limitante inferior descrito pela Proposição 2 foi utilizado em ambos os algoritmos. As tabelas 1 e 2 exibem os resultados computacionais. O tempo máximo, mínimo e a média para a execução de cada algoritmo foi calculado para cada valor de n. A unidade de tempo utilizada é o milissegundo (ms). Calculamos também o máximo e a média do número de nós gerados por cada algoritmo, além da quantidade de testes na qual a solução ótima foi obtida no primeiro nó da árvore.

n	Algoritmo 1					
	Número Nós		Tempo Execução			Solução no Primeiro Nó
	Média	Max	Min	Média	Max	
20	6,05	90	0	1,17	16	51
40	14,57	263	0	8,96	124	50
80	23,17	761	0	67,70	1840	54
150	10,44	136	15	161,55	1670	53
200	13,66	250	31	498,59	10640	56

Tabela 1: Resultados Computacionais do Algoritmo 1

n	Algoritmo 2					
	Número Nós		Tempo Execução			Solução no Primeiro Nó
	Média	Max	Min	Média	Max	
20	3,35	86	0	1,25	30	59
40	3,75	40	0	2,33	31	57
80	5,70	143	0	8,18	125	59
150	3,47	55	15	31,40	358	64
200	4,82	77	31	72,81	406	61

Tabela 2: Resultados Computacionais do Algoritmo 2

O número de nós produzidos pelo Algoritmo 1 variou entre 1 e 761 e no Algoritmo 2, o

número de nós alterou entre 1 e 143 . Em todos os valores de n, a média de nós gerados pelo Algoritmo 2 foi inferior a do Algoritmo 1.

A solução ótima foi obtida no primeiro nó da árvore para 0,66% dos testes no Algoritmo 1 e 0,75% no Algoritmo 2. Isto indica que ambos os Algoritmos reconheceram rapidamente, como ótimo, o sequenciamento obtido pelo Algoritmo de Schrage.

O tempo de execução do Algoritmo 1 para resolver os problemas testes variou entre 0 e 10,64 segundos. Para o Algoritmo 2, a variação de tempo ficou 0 e 0,406 segundo. Exceto para o caso de $n = 20$, a média de tempo para o Algoritmo 2 foi inferior a do Algoritmo 1. A diferença significativa entre os tempos nos dois Algoritmos se deve a fato de que o Algoritmo 1 gerou, em média, mais nós na árvore de busca e a cada nó gerado um tempo adicional foi incorporado em decorrência da execução do Algoritmo de Schrage.

5 Conclusão

Neste trabalho comparamos o desempenho computacional de dois algoritmos para o problema de minimização do *makespan* no sequenciamento de tarefas em uma única máquina. Cada tarefa tem um tempo de processamento, um tempo de entrada no sistema e um tempo em que a tarefa fica no sistema após a sua conclusão. Para resolver o problema, utilizamos o método *Branch* e *Bound* e as estratégias utilizadas por Carlier [2] (Algoritmo 1) e Grabowski, Nowicki e Zdrzalka [5] (Algoritmo 2). Ambos os algoritmos determinaram a solução ótima para os 400 problemas testes. Foram analisados para cada algoritmo o número de nós gerados na árvore de busca e o tempo de processamento. O Algoritmo 2 apresentou os melhores resultados, tanto para o número de nós, quanto para o tempo de processamento. Em ambos os Algoritmos observou-se a repetição de muitos sequenciamentos na árvore de busca, além de muitos sequenciamentos com o mesmo limitante inferior. Uma possibilidade para melhoria do algoritmo seria a utilização do procedimento de *Backtracking* na árvore de busca e de novas formas de atualização de r_i e q_i.

Referências

[1] P. Brucker, B. Jurisch, B. Sievers, A branch and bound algorithm for the job-shop scheduling problem, *Discrete Applied Mathematics*, 49, (1994), 107-127.

[2] J. Carlier, The one-machine sequencing problem, *European Journal of Operational Research*, 11, (1982),42-47.

[3] J. Carlier, E. Pinson, An algorithm for solving the job-shop problem, *Management science*, 35, (1989), 164-176.

[4] M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", Freeman, New York, 1979.

[5] J. Grabowski, E. Nowicki, S. Zdrzalka, A block approach for single-machine scheduling with release dates and due dates, *European Journal of Operational Research*, 26, (1986), 278-285.

[6] J. Grabowski, J. Pempera, The permutation flow shop problem with blocking. A tabu search approach, *Omega*, 35, (2007), 302-311.

[7] J. R. Jackson, Scheduling a production line to minimize maximum tardiness, *Management Science Research*, 43, (1955).

[8] J. K. Lentra, "Sequencing by enumerative methods", Mathematisch Centrum Amsterdam, (1976).

[9] J. K. Lenstra, A. R. Kan, P. Brucker, Complexity of machine scheduling problems, *Econometric Institute of the Erasmus University*, (1977).

[10] C. F. Liaw, A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem, *Computers and Operations Research*, 26, (1999), 679-693.

[11] J. F. Muth, G. L. Thompson, "Industrial Scheduling", Prentice Hall, New Jersey, 1963.

[12] L. Schrage, Solving resource-constrained network problems by implicit enumeration non-preemptive case, *Operations Research* , 18, (1970), 263-278.

