Earliness Penalties on A Single Machine Subject To No Tardy Jobs

Tariq S. Abdul-Razaq and Manal G. Ahmed Al-Ayoubi
Mathematic Dept College of science Al- Mustansiriyah University
Received 6/10/2008 – Accepted 31/12/2008

الخلاصة

جدولة مسألة ناقشنا ، الدراسة هذه في n إيجاد هو هدفنا .واحدة ماكنة على المستقلة الأعمال من
متأخرة أعمال هنالك يكون لا بحيث الكلية، التبكير جزاءات لتصغير التقريبية والحلول الأمثل الحل.
(To minimize total earliness penalties subject to no tardy jobs)
الدنيا القيود من مختلفة أنواع اقترحنا لقد (LB) الأحسن الأدنى القيد وهو جديد احدها للمسألة
والتقيد التفرع خوارزمية في واستخدم (BAB) اشتقاقه الجديد الأدنى القيد وهذا . المثلى الجدولة إيجاد لغرض
سمث طريقة استخدام من عليها نحصل والتي للجدولة (المبكرة الكلفة) الهدف دالة أرخاء على يعتمد
التراجعية.
النوع من هي مسألتنا إن بما ، الأمثل الحل إلى تؤدي والتي للمسألة الخاصة الحالات بعض برهنا لقد
الصعب (NP-hard)، مثل تقريبية طرائق باستخدام قمنا (DM,APIM,TTHM) تربط جديدة وطريقة
(DM) مع (APIM) من كبيرة مجموعة على الطرق هذه أداء تقيم وعند . الأمثل الحل من قريب حل لإيجاد
إن وجدنا الاختيارية المسائل (TTHM) ثم كفاءة أكثر (APIM) وأخيرا (DM) .
الواحدة الماكنة في التبكير لجزاء الأمثل الحل من قريبة حلول لإيجاد تقريبية طرق أربع قدمنا لقد
للمسالة.

ABSTRACT

This study discusses the problem of scheduling n independent jobs on a single machine. The aim of the study is to find the optimal and near optimal schedules that minimizes the total earliness penalties subject to no tardy jobs.
The study proposes different lower bounds; the new one which is the best lower bound (LB) for the problem is based on a relaxation on earliness cost to be used in a branch and bound algorithm in order to find an optimal schedule. The derivation of this new LB depends on the earliness cost(which is relaxed) for the sequence obtained by using Smith's backward procedure. We proved some special cases which led to optimal solution, since our problem is NP-hard, some local search methods are also applied such as: Descent method (DM), adjacent pairwise interchange method (APIM), tree type heuristic method (TTHM) and proposed a new method that linking descent method with adjacent pairwise interchange method. We evaluate the performance of these methods on a large set of test problems. The results, show that (TTH) method is the best, followed by (API) and (DM) methods. Also we present four simple heuristic methods to find near optimal solutions for earliness penalties on a single machine subject to no tardy jobs.
Key words : Single machine , Earliness , No tardy job , Lower bound , Near optimal solution

INTRODUCTION

In this paper we are studying the earliness penalties on a single machine subject to no tardy jobs. This kind of scheduling problems found in situations where tardiness cost is extremely high compared to earliness cost or when tardy jobs are prohibited. This class of problems is also found in production situations that involve many customers and the customers generally operation in a JIT (just in time) environment where there is no tolerance for tardiness. In this problem, if a job is completed earlier than its due date, then the earliness cost is incurred for holding the job until its delivery time. A job can not be delivered to a customer before or after its due dates. It must be shipped to the customer exactly on its prespecified due date [1]. Ideally, the production is shipped to the customers on its time in this case, there is no earliness cost and if it is completed before its due date then a holding cost (penalty) w_i per unit time is incurred while a job i wait for shipment at time d_i.The aim of this problem is that, the jobs are completed on their due dates or near their due dates in order to make the earliness amounts suitable enough to reduce the holding cost (earliness penalties) of the jobs.

There are few studies of earliness penalties on a single machine, (i.e., $1//\sum_{i=1}^{n} W_iE_i$ problem), because this problem is equivalent to tardiness penalties on a single machine (i.e, $1//\sum_{i=1}^{n} E_iW_i \approx 1//\sum_{i=1}^{n} T_iW_i$). It is well-known that $1//\sum_{i=1}^{n} W_iT_i$ is NP-hard problem and most researches have studies the tardiness penalties on a single machine, for more details (see [2], [3]).

The other types of studies are about the sum of earliness and tardiness penalties on a single machine, the inclusion of both earliness and tardiness costs in the objective function is compartible with the philosophy of just in time production, which emphasizes producing goods only when they are needed [4], for more details (see [5], [6]).

For the above mentioned reasons we have been urged to study the total earliness penalties.

The Mathematical Model of the Problem and the Structure of Optimal Solution

Our scheduling problem can be states as follows:

Let N = {1,2,……,n} be the set of independent jobs (i.e., no precedence constraint are on jobs) which are processed on a single machine. The jobs have to be scheduled without preemption on the machine, where the machine can process one job at a time (i.e., the machine cannot

process two jobs at the same time). The machine and the jobs are assumed to be available from time zero and the machine idle time is not allowed, the assumption that no machine idle time is allowed reflects a production setting where the cost of machine idleness is higher than the early cost incurred by completing any job before its due date, or the capacity of the machine is limited when compared with its demand, so that the machine must indeed be kept running [4].

Each job has processing time P_i, weight W_i and due date d_i (the ideal time that the job be completed on it) and they are all known in advance.

Given a schedule $\sigma = (\sigma(1),\ldots,\sigma(n))$, then for each job i we calculate the completion time $C_{\sigma(i)} = \sum_{j=1}^{i} P_{\sigma(j)}$, where $\sigma(i)$ denote the position of job i in the schedule σ.

Let S be denote the set of all feasible sequence where $\sigma \in S$.

The objective is to find the processing order of the jobs that minimizes the total earliness penalties in the following problem (P).

$$\underset{\sigma \in S}{Min} \sum_{i=1}^{n} w_{\sigma(i)} E_{\sigma(i)}$$

s.t.

$$C_{\sigma(i)} \leq d_{\sigma(i)} \quad \forall\ i = 1,2,\ldots,n \qquad (P)$$
$$E_{\sigma(i)} = d_{\sigma(i)} - C_{\sigma(i)},\ i = 1,2,\ldots,n$$
$$E_{\sigma(i)} \geq 0 \quad \forall\ i = 1,2,\ldots,n$$

Using the three field classification suggested by Graham et al.

[7], the problem (P) is denoted by $1 / C_i \leq d_i / \sum_{i=1}^{n} w_i E_i$. In this type of arbitrary due dates the problem of minimizing total earliness with no tardy jobs is proved to be NP-hard [8].

We first present these definitions:

Definitions and Some Results

A sequence usually corresponds to a permutation of the job set or the order in which jobs are to be processed on a given machine.

A schedule refers to an allocation of jobs within a more complicated setting of machines, which could a low for preemptions of jobs by other jobs that are released at later points in time and start / completion time [9].

It should be noted that in this study there is no difference between a sequence and a schedule.

A schedule is called **feasible schedule,** in this study, if no tardy jobs exists in this schedule (i.e, $C_i \leq d_i \; \forall$ i= 1,2,….,n).

It is clear from the definition of feasible schedule, in all feasible schedules of $1 / C_i \leq d_i / \sum_{i=1}^{n} w_i E_i$ problem, we have

1- $T_{max} = \max_i \{T_i\} = 0$, where $T_i = \max_i \{C_i - d_i, 0\}$

2- The total weighted number of late jobs is zero.

Theorem (1) LWPT rule (i.e., sorting the jobs in non- increasing order of the ratio P_i/w_i) gives the minimum $\sum_{i=1}^{n} w_i E_i$ for the problem $1 / C_i \leq d_i / \sum_{i}^{n} w_i E_i$

$_{i}\Sigma_{=1}$

.

Proof :

Let N= {1, 2,…, n} be the set of jobs to be scheduled. After we scheduled the jobs of N, we have feasible sequences and non-feasible.

Let S = {s1,s2,……,sr}, r ≤ n! be the set of feasible sequences to

1 /Ci ≤ di / $w\ E_i$

n

i

$_{i}\Sigma_{=1}$

problem, after we omit the non-feasible sequences.

Among all feasible sequences we shall show that the feasible sequence which is ordered by LWPT rule minimizes $w\ E_i$

n

i

$_{i}\Sigma_{=1}$

.

Let σ = (σ (1), σ (2),……, σ (n)) be the sequence obtained by LWPT rule.

Since Σ =

∈ =

$w\ E\ _i$

n

i

i

S

( )

1

min σ ( ) σ

σ

min { ( )(( ) ( ))}

$_{1\,i\,i}C\ _i$

$s\,d$

n

$_i\,W$σ σ σ

σ

− Σ=

∈

min{ ( ) ( ) ( ) ( )}

1 1 *i i i* C_i
S
n n
i i w_σ d_σ w_σ σ
σ
Σ −Σ
∈ = =
=

It is clear that **R.H.S.** is minimized by LWPT rule, since the first term is constant and the second term *i* C_i
n
i w ( ) ( )
1
σ σ Σ=
is maximized by LWPT rule.

Corollary (1)

For the 1 /C$_i$ ≤ d$_i$/ w E_i
n
i
i Σ=1
problem SWPT rule gives the maximum
w E_i
n
i
i Σ=1
.

Proof:-

It is clear from theorem (1), since LWPT rule gives the minimum
w E_i
n
i
i Σ=1
, and SWPT rule is the inverse of LWPT rule.

Hence, the SWPT rule gives the maximum w E_i
n
i
i Σ=1
.

The following result shows the structure of optimal schedules of our scheduling problem (P).

Theorem (2)

For the $1 / C_i \le d_i / \sum_{i=1}^{n} w_i E_i$ problem, if LWPT rule is not feasible, then there exists at least one feasible schedule with $\sum_{i=1}^{n} w_i C_i$ as large as possible is optimal.

Proof:

After we scheduled the jobs of N, we get feasible schedules with $C_i \le d_i$, i = 1,2,…,n and non-feasible schedules with $C_i > d_i$ for at least one job i (i=1,2,…,n).

Let $S = \{s_1, s_2, \ldots\ldots, s_r\}$, $r \le n!$ be the set of feasible schedules,our aim is to minimize $\min_{S_j \in S}\{\sum_{i=1}^{n} w_i E_i = \sum_{i=1}^{n} w_i (d_i - C_i)\}$,

where $s_j \in S$ (j=1,2,…,r). We look for a schedule which maximizes $\sum_{i=1}^{n} w_i c_i$, because $\sum_{i=1}^{n} w_i d_i$ is constant.

From theorem (1) LWPT schedule maximize $\sum_{i=1}^{n} w_i c_i$

, but this schedule is not feasible. Hence, the first feasible schedule $s_j \in S$ $(j=1,2,\ldots,r)$ after LWPT schedule that maximizes $\sum_{i=1}^{n} w_i C_i$ is optimal for $\sum_{i=1}^{n} w_i E_i$ problem.

The following result, determines the lower and the upper bounds for the total weighted earliness.

Theorem (3)

If **E*** = min { $\sum_{i=1}^{n} w_i E_i$ }, then

$$\sum_{i=1}^{n} w_i (d_i - C_i) \le \mathbf{E^*} \le \sum_{i=1}^{n} w_i (d_i - c'_i)$$

where $\sum_{i=1}^{n} w_i C_i$ and $\sum^{n} w\, c'_i$

i

$_{i}{}' \sum_{=1}$

are obtained by LWPT and SWPT rules respectively.

Proof:

Suppose the jobs are order (1,2,….,n). In LWPT schedule, if this schedule is feasible then it is optimal and exact lower bound on E* (obtained by using theorem (1) and E* = $(\ )$

$_{1}\, {}_{i}C_{i}$

n

i

$_{i}d\ w - \sum_{=}$

.

If LWPT schedule is not feasible, then a lower bound on E* is obtained by using theorem (2).

For upper bound, suppose the SWPT schedule (1,2,…,n) gives $w\ E_{i}$

n

i

$_{i}\sum_{=1}$

$= (\ )$

$_{1}w\ d\ c\ _{i\,i}$

n

i

$\sum\ _{i} - '$

$=$

which yield the upper bound if SWPT is feasible.

E* $\leq w\ E_{i}$

n

i

$_{i}\sum_{=1}$

$= (\ )$

$_{1}w\ d\ c\ _{i\,i}$

n

i

$\sum\ _{i} - '$

$=$

If SWPT schedule is not feasible schedule, then the first feasible schedule which minimizes $w\ c\ _{i}$

$$\sum_{i=1}^{n} {}_i{}'$$

yields the upper bound for E*.

It is clear that theorem (3) is used to derive lower and upper bounds on the minimum total weighted earliness.

Remark (1): For the special case of unit weights, (i.e., $w_i = 1$ for $i = 1,2,\ldots,n$) then the resulting problem becomes $1 / C_i \leq d_i / \sum_{i=1}^{n} E_i$ and all the results above are true for it.

Lemma (1):-

In an optimal schedule for $1/C_i \leq d_i / \sum_{i=1}^{n} w_i E_i$ problem, the last job i with $d_i \geq T = \sum_{j=1}^{n} p_j$ and with smallest P_i / w_i, $i = 1,2,\ldots,n$.

Proof:-

1. If LWPT schedule is feasible schedule, then the last job i with $d_i \geq T = \sum_{j=1}^{n} p_j$ with smallest P_i / w_i $i = 1,2,\ldots,n$ by theorem (1).
2. If LWPT schedule is not feasible schedule, then by theorem (2) the first feasible schedule after LWPT schedule is optimal. In this schedule, if there exists more than one job with $d_i \geq T = \sum_{j=1}^{n} p_j$, then the job with smallest P_i/w_i should be the last job.

***Remark (2)*:** We notice that in general when the LWPT schedule is not feasible schedule, the optimal schedule consists of subschedules each of which is ordered by LWPT rule .

A special Case

In this section, we introduce three special cases for $1 / C_i \leq d_i / \sum_{i=1}^{n} w_i E_i$ problem. In the first case the due date is common for all jobs, but in the second case the processing time is common for all jobs and the $w_i = w$. In the third case we present the ideal solution. Now we are going to present these cases.

1- **If $d_i = d = \sum_{j=1}^{n} p_j$** (i.e., the due dates of all jobs are constant and equal to the sum of processing times of all jobs), then the $1/C_i \leq d_i = d/ \Sigma w_i E_i$ problem has optimal solution obtained by theorem (1). Notice that, in this case we have: -

- All sequences are feasible sequences.
- In all feasible sequences, the last job completed on its time and the other (n -1) jobs completed before their due dates, this means that all jobs in all feasible sequences are early.

2- If $P_i = P$, ($w_i = w$), $\forall$ i = 1, 2,…,n.

Then the first feasible sequence is optimal for $1/c_i \leq d_i/ \sum_{i=1}^{n} w E_i$ problem. This is clear, since $\sum_{i=1}^{n} w E_i$ is constant and is the same for all feasible schedules.

3- **The ideal solution**, if all the jobs completed exactly on their due dates (i.e., $C_i = d_i \; \forall$ i=1,…..,n), then for this case there exists no earlier cost (i.e., $w_i E_i = 0 \; \forall$ i= 1,…..,n and $\sum_{i}^{n}$

$w_i E_i$ 1

= 0)

Derivation of Lower Bounds

In this section, we are going to show the methods of derivation

the lower bound for $1/c_i \leq d_i/ \sum_{i=1}^{n} w_i E_i$

problem.

Relaxation of Object

We can calculate a lower bound for earliness penalties on a single machine, by using a relaxation of object as follows:

- Sequencing the jobs of the set N = {1,2,…,n} to get a feasible schedule (1,2,….,n).
- Let $u_i \leq w_i \; \forall$ i=1,….,n

Then

$$LB(u) = \text{Min}\,\{\sum_{i=1}^{n} u_i (d_i - C_i)\} \leq \text{Min}\,\{\sum_{i=1}^{n} w_i E_i\}$$

where u = (u_1,…,u_n).

Dynamic Programming State Space Relaxation (DPSSR)

This method was developed by Christfides et al. for various routing problem and applied first in scheduling by Abdul-Razaq and Potts [10].

We can use this method to get a lower bound (LB) for the problem of scheduling jobs on a single machine to minimize total cost, for example to get a lower bound for $1/c_i \leq d_i/ \sum_{i=1}^{n} w_i E_i$

problem.

Let S ⊆ N be an arbitrary subset of jobs. Let f(S) be the minimum total cost when the jobs of S are sequenced in the first S positions in the sequence. The objective is to find f (N) by solving the recursion equations.

$$f(S) = \text{Min } \{f(S - \{i\}) + g_i(\sum_{j\in S} p_j)\} \;......(1)$$

where $f(\varphi) = 0$ and $g_i(t)$ is the cost of job i if it completed at time t, $g_i(t) = \infty$ if $t > d_i$ (i.e., the schedule is not feasible).

Solving recursion equations (1) is equivalent to find the shortest path in the state space graph G in which vertices correspond to subsets S and the length of an arc directed from vertex S-{i} to vertex S is $g_i(\sum_{j\in S} p_j)$, $i \in S$. Clearly there are 2^n vertices in G. Let $T = \sum_{i=1}^{n} p_i$ and

$$g_i(t) = \text{Min } \{w_i(d_i - t)\}.$$

To find lower bound by DPSSR, our object to find $f_0(T)$ from the recursion equations

$$f_0(t) = \text{Min } \{f_0(t - p_i) + g_i(t)\} \;......(2)$$

where $f_0(t) = \infty$ if $t < 0$, $g_i(t) = \infty$ if $t > d_i$ (i.e., the schedule is not feasible).

The state space graph for recursion (2) has vertices 0,1,….,T and for each job i ($i \in N$) has an arc directed from vertex t-p_i to vertex t for $t = p_i, p_{i+1}, ..., T$. Then we may regard $f_0(t)$ the minimum cost of scheduling jobs in the time interval [0, t].The computation of $f_0(T)$ requires O(nT) time [10].

The above equation (2) gives a tight lower bound (LB), but the disadvantage of this method is that, it is suitable for small number of jobs but as the number of jobs becomes larger (i.e., for 20 jobs and more), this method lasts much more time and this reason makes this method unpractical and unusable for large number of jobs .

A Simple Lower Bound

The suitable, simple and efficient lower bound (LB) with respect to the previous lower bounds is obtained by theorem (1) or (3). It is clear that for the special case $w_i = 1$ for each i, the lower bound can be obtained for the unweighted case.

Derivation of the Best Lower Bound

In this section, we shall be interested in deriving a new lower bound on the optimal solution.

Let S be the set of unsequenced jobs, σ be an initial partial sequence of jobs.

Suppose we have the problem (P1)

$$\text{Max } \Sigma C_i$$

$$\text{s.t. } C_i \leq d_i \text{ for all } i \quad (P1)$$

Then at least one feasible solution to (P1) , can be relaxed to be a lower bound for $1//\Sigma E_i$ problem. We derive our new lower bound by using modified Smith backward procedure to solve (P1), as follows

Step 1 Set T = $\sum_{i=1}^{n} P_i$, N ={1,2,…,n}, k = n.

Step 2 Find a job j*: $P_{j^*} \leq P_j$, $d_{j^*}, d_j \geq T$, j, j* $\in$ N, (break ties by selecting the job j* with largest due date) assign job j* to position k.

Step 3 Set T= T-P_{j^*}, N=N-{j*}, k=k-1, if k=0 stop, else go to (2).

It is clear that this procedure gives a feasible schedule. Suppose this feasible schedule is (1,2,…,n). Then we calculate the earliness E_i' , (i=1,2,…,n) for this feasible schedule and relax this E_i' in this schedule by:

$E_i = \min\{P_i, E_i'\}$, for each i in this relaxed problem to obtain LB = $\sum_{i=1}^{n} E_i$ which is less than the optimal solution.

Hence, the lower bound LB(S) is given by:

$LB(S) = \sum_{i\in\sigma} E_i + \sum_{i\in S} E_i$, where S is the set of unsequence jobs.

It is clear that for the special case when σ is empty, we have the initial lower bound $LB(S) = \sum_{i\in S} E_i$.

Branch and Bound Algorithm

In this section, we shall describe our branch and bound algorithm which is widely used in scheduling problems.

Derivation of an Upper Bound

The first method to derive the upper bound by modification of Smith backward procedure as follows:

Step 1 Set $T= \sum_{i=1}^{n} P_i$, $N =\{1,2,\ldots,n\}$, $k = n$.

Step 2 Find a job j^*: $P_{j^*} \geq P_j, d_{j^*}, d_j \geq T$, $j, j^* \in N$ then assign job j^* to position k.

Step 3 Set $T= T-P_{j^*}$, $N=N-\{j^*\}$, $k=k-1$, if $k=0$ stop, else go to (2).

Order the jobs according to the procedure above, let the schedule is (1,2,..,n) and calculate $\sum_{i=1}^{n} E_i$ for this feasible schedule, then set $UB1 = \sum_{i=1}^{n} E_i$.

The second method to derive the upper bound by EDD rule (i.e., $d_1 \leq d_2 \leq \ldots \leq d_n$) and calculate $\sum_{i=1}^{n} E_i$

for this feasible schedule, then UB2 = $\sum_{i=1}^{n} E_i$.

UB= min {UB1,UB2} which is the initial upper bound at the root node of the search tree.

Derivation of Lower Bound

The lower bound which is used in branch and bound (BAB) is the same as in section (5-4).

The Forward Branching

At any level, we calculate the lower bound LB(S) given in section (5-4) for each node (by calculate the cost of sequence jobs and the cost of unsequence jobs in a set S). If a node has a lower bound greater than or equal the current upper bound already computed, then this node is discarded, and we chose the active node (a node with smallest lower bound) to branch from it. The BAB method continuous in the same way by using a forward branching rule.

Whenever a complete sequence is obtained, this sequence is evaluated and the upper bound (UB) is altered if the new value is less than the old one. The procedure is repeated until all nodes have been considered.

The Use Of Local Search Method

In this section, we are going to describe the methods of local search that we have used in this study.

Adjacent Pairwise Interchange Method (APIM)

This method defined by a pair interchange operators which interchange elements (jobs) at positions (i) and (i+1) for a given sequence (i=1,2,….,n-1) [11].

Now we are going to describe the steps of (APIM).

• Initialization

The initial current solution is obtained from the derivation of upper bound that we use in (BAB) algorithm to be the initial upper bound with its current sequence $\sigma = (\sigma(1),\ldots\ldots,\sigma(n))$.

• Neighbor generation

We are going to improve the sequence σ by chosen job i arbitrary and interchange the two adjacent jobs i, i+1 ($1 \leq i \leq n-1$) (i.e., the positions of two adjacent jobs σ (i), σ (i +1), $1 \leq i \leq n-1$ are transposed).

The new sequence σ * that will be obtained with its objective function value AUB.

The second API2 by improving σ by chosen arbitrary the pair (i, i+1), ($1 \leq i \leq n-3$) and interchange with (j, j+1), ($i+2 \leq j \leq n-1$). The new sequence σ * that will be obtained with its objective function value A2UB.

- **Evaluation**

If AUB < UB then the two adjacent jobs are left in their new positions in σ *, otherwise the two adjacent jobs are replaced in their original positions.

After that the procedure repeated from step (Neighbor generation).

The evaluation step for the second improving is the same as the above evaluation but instead of AUB we use A2UB.

- **Termination step**

This method is terminated after (1500) iterations.

Descent Method (DM)

We are going to use this method in both cases (Insert neighbor and Swap neighbor) as follows:

- **Initialization**

This step is the same as initialization of (APIM).

- **Neighbor generation**

In this step, the first neighbor is insert neighbor then $\sigma' = (\sigma'(1),\ldots, \sigma'(n))$ is a feasible neighbor (if $C_i \leq d_i \forall i$) of the current solution generated by chosen randomly job i ($1 \leq i \leq n$) from σ and, this job i shift arbitrary to the left. The objective function value of insert neighbor is (IUB).

The second neighbor is swap neighbor, the feasible neighbor (if $C_i \leq d_i \forall i$) $\sigma' = (\sigma'(1),\ldots, \sigma'(n))$ of the current solution generated by chosen randomly two jobs σ (i), σ (j), ($i \neq j$), ($1 \leq i, j \leq n$) from σ (not necessary adjacent) and interchange them. The objective function value of swap neighbor is (SUB).

The third neighbor is insert-swap neighbor, at each step the current solution is improving by insert neighbor and this sequence is then improved by swap neighbor. The objective function value of insert-swap neighbor is (In_SwUB).

The forth neighbor is swap2 neighbor, the feasible neighbor (if $C_i \leq d_i \forall i$)

$\sigma'_1 = (\sigma'_1(1),\dots,\sigma'_1(n))$ of the current solution generated by chosen randomly two jobs i, j, ($i \neq j$ & $i < j$), ($1 \leq i \leq n-3$ & $3 \leq j \leq n-1$) from σ (not necessary adjacent) and interchange (i, i+1) with (j, j+1). The objective function value of swap neighbor is (Swap2UB).

• **Acceptance test**

In this step, we are going to test whether to accept σ' or retain to σ for the previous neighbors, as follows:

a- If (IUB < UB) then σ' replace σ as the current solution and we set UB = IUB then we go to step (Neighbor generation).

b- Otherwise (i.e., IUB $\geq$ UB) then σ retain as the current solution, and we return to step (Neighbor generation).

The substeps (a), (b) for swap neighbor are the same for insert neighbor but instead of (IUB) we put (SUB).

The substeps (a), (b) for insert-swap neighbor are the same for insert neighbor but instead of (IUB) we put (In_SwUB).

The substeps (a), (b) for swap2 neighbor are the same for insert neighbor but instead of (IUB) we put (Swap2UB).

• **Termination test**

After (1500) iterations, we terminate this algorithm for insert,swap, insert-swap and swap2 at local minimum.

Another neighbor is linking API2 with swap, the steps are the same as the previous steps but Neighbor generation is at each step we improving the current solution by API2 and the resulting sequence improving by swap neighbor, the objective function value of this neighbor is AP-Sw. The last neighbor is linking API2 with swap2 which is the same as API2 with swap but the improving sequence obtained from API2 is improving by swap2 neighbor, the objective function value of this neighbor is AP-Sw2.

The Use of Tree Type Heuristic Method (TTHM)

In a branch and bound procedure (BAB), a near optimal solution is obtained if some of the possible optimum partial schedules have not been explored. This method is defined as the tree type heuristic (TTH). We start with this method as we have started with branch and bound (BAB) to use the upper bound at the top of the search tree in the same way as we used in section (6-1) in BAB.

At each level, we calculate the (LB) for the feasible nodes in order to specify from which node to branch in according to lowest value of the lower bound among the lower bounds of all nodes. We continue in the same way until we reach the final level by using a forward branching without backtracking. We evaluate the complete sequence which is obtained. The upper bound is changed if the new value (TUB) is less than the old one. (i.e., TUB < UB, then set UB = TUB).

***Heuristics for* $1/C_i \leq d_i/\ \Sigma w_i E_i$**

Heuristic (1)

Step 1

Order the jobs by LWPT rule, if it is feasible then it is optimal. If LWPT schedule is not feasible (i.e., there exists at least one job s.t. $c_i > d_i$ for some i), then find a job j such that $d_j \geq T = \sum_{i=1}^{n} p_i$ if there is a tie choose j with smallest P_j / w_j (by lemma 1) and schedule j in the last position.

Step 2

Order (n-1) jobs by LWPT rule, if it is feasible then it is optimal.

Step 3

Otherwise (i.e., (n-1) jobs are not feasible by LWPT), then the first tardy job i interchange with one of the early jobs j that precedes i such that:

$$d_j \geq T = \sum_{k=1}^{i} p_k$$

and with smallest P_j / w_j ……… (3)

as well as that job i and job j are early. Repeat this process until interchange all the tardy jobs with the early jobs and reach the corresponding feasible schedule, then calculate UB1 = $\sum^{n}$

i

$w_i E_{i\,1}$

for this schedule.

Step 4

If there exists a tardy job i such that step (3) is not satisfied, then find an early job j with $d_i \leq d_j$ and with smallest P_j / w_j, then set i precedes j, such that the both jobs i and j are early.

Note

For the LWPT schedule if there exists two jobs i and j such that:

- If p_i / w_i (for tardy job) $\geq p_j / w_j$ (for early job) and
- $d_i \leq d_j$, then job i precedes job j such that job i and job j are early.

If only the condition ($d_i \leq d_j$) is satisfied then job i precedes job j such that job i and job j are early.

Heuristic (2)

Step 1

If LWPT schedule is not feasible then for this LWPT schedule find the set J_1 of early jobs

$J_1 = \{i \in N: d_i \geq T$, where $T = \sum_{=}$

i

j

$_j p$

1

, i= 1,2,…,n}, sequence the jobs of J_1 in LWPT.

Step 2

Find $J_2 = \{i \in N, i \notin J_1\}$, sequence the jobs of J_2 in LWPT rule, if J_2 is feasible and (J_2, J_1) is feasible then calculate UB2 $= \sum_{=}$

n

i 1

$_{i\,i}$WE for (J_2, J_1).

Step 3

If J_2 is not feasible or (J_2, J_1) is not feasible then find the tardy job $j \in J_2$ with $C_j > d_j$ and compare with the early jobs that precede it and make use of the note above until we find the first feasible schedule then calculate UB2 $= \sum_{=}$

n

i 1

$_{i\,i}$WE for this feasible schedule.

Heuristic (3)

Step 1

Calculate $T = \sum_{j=1}^{n} p_j$ and find the job i such that $d_i \geq T$ with smallest p_i / w_i then set job i in position n (i.e., i = n).

Step 2

Set $T = T - p_i$, n=n-1 and find a job i with $d_i \geq T$ and with

smallest p_i / w_i then set i=n. Repeat this process until the first early job, when i =1 and Calculate UB3= $\sum_{i=1}^{n} W_i E_i$ for this feasible schedule.

Heuristic (4)

Step 1

Order the jobs in LWPT, if the resulting schedules is feasible, then it is optimal schedules. Otherwise go to step 2.

Step 2

Let $T = \sum_{i=1}^{n} p_i$, N={1,2,….,n}.

Step 3

Find the set $J_1 = \{i \in N: d_i \geq T\}$ then order the jobs of J_1 in LWPT order (in the last unfilled positions) to construct the final sequence J_2. Let $TJ_1 = \sum_{i \in J_1} p_i$, set $T = T - TJ_1$, $N = N - J_1$.

If $N \neq \{\varphi\}$ repeat step (3), otherwise go to step (4).

Step 4

$UB4 = \sum_{i \in J_2} W_i E_i$

, where J_2 is the sequence obtained from the subsequences of J_1.

Computational Experience

Tests Problems

The (BAB) algorithms and (TTHM) were tested by coding them in Microsoft Fortran Power station and the local search was tested by coding it in Math lab 6.5 and running on a Pentium IV at 1700 Mnz, with Ram 128 MB computer.

Thirty-six random test problems were generated for each value of n. The size of the test problems used are n = 10, 15, 20, 25, 30, 40, 50, 75, 100, 150. The processing times P_i for $i \in N = \{1, 2, \ldots, n\}$ is generated by randomly selecting integers from the uniform distribution [1, 10]. Due dates are generated based on different values ($D \in \{0.3, 0.6, 0.9\}$ and $E \in \{D+0.3, D+0.6, D+0.9\}$). The due dates d_i $(i=1,2,\ldots,n)$ are uniformly distributed integers from the interval [DT, ET] where $T = \sum_{i=1}^{n} P_i$.

We use (BAB) algorithm that described in section (6) to find the optimal solutions of all 10-jobs, 15-jobs, 20-jobs, 25-jobs, 30-jobs and 40-jobs, which are obtained with reasonable limits on computation

time. Problems that are not solved to optimality because their nodes were more than (10000000) nodes and lasted very long time, the best solution for these unsolved problems found by any of our local search methods were used instead, of the BAB algorithm.

The local search methods that are described in section (7) were applied and compared to all test problems.

Comparative Computation Results for BAB Method.

In this section, we are going to compare the results which we have reached in BAB method. In table (1) we compare the optimal solution with the upper bound (UB), the lower bound (LB) and we show the number of nodes for n = 30 jobs.

In table (2) we compare the number of nodes for the solved problems in BAB for n = 10, 15, 20, 25, 30, 40. Where NS1 is the number of problems solved that require not more than (200) nodes, NS2 is the number of problems solved that require over (200) nodes and not more than (2000) nodes, NS3 is the number of problems solved that require (2000) nodes and not more than (20000) nodes, NS4 is the number of problems solved that require over (20000) nodes. Finally, NU explains the number of unsolved problems (that require more than 10000000) nodes.

Table-1: comparative computation results for n=30 for BAB

.No	LB	.Opt	UB	NOD
1	126	173	212	67009
2	115	141	243	14198
3	123	181	228	152641
4	115	148	185	22483
5	110	143	208	18593
6	95	135	162	279865
7	142	183	217	119904
8	102	148	196	70076
9	122	167	194	150499
10	112	151	192	156808
11	127	178	215	88776
12	117	156	206	29638
13	152	192	237	44573
14	114	146	169	30491
15	146	198	237	152318
16	118	167	193	240255
17	151	195	231	36672
18	133	183	257	190062
19	122	159	194	90254
20	100	154	162	121324
21	105	169	202	575149
22	123	157	195	25962
23	126	162	231	34167
24	112	152	203	68666
25	111	143	202	13123
26	122	162	193	78311
27	132	167	214	150241
28	124	176	211	210743
29	128	175	200	230380
30	140	176	237	37801
31	91	111	161	11625
32	130	177	246	155437
33	116	159	225	13279
34	119	172	211	104063
35	146	197	228	177564
36	132	168	242	43018

No : Test problem number.
LB : Lower bound.
Opt. : Optimal solution.
UB : Upper bound.
NOD: Number of nodes in branch and bound method.

Table -2: comparative results for number of nodes for BAB Method

n	NS1	NS2	NS3	NS4	NU
10	34	2	-	-	-
15	2	33	1	-	-
20	-	12	22	2	-
25	-	-	20	16	-
30	-	-	5	31	-
40	-	-	-	28	8

NS1: Number of problems solved that require not more than (200) nodes.
NS2: Number of problems solved that require over (200) nodes and not more than (2000) nodes.
NS3: Number of problems solved that require over (2000) nodes and not more than (20000) nodes.
NS4:Number of problems solved that require over (20000)nodes.
NU: number of unsolved problems(require more than 10000000) nodes.

Comparison between the Optimal Solution and the Solutions Obtained By the Local Search Methods.

This section shows the efficiency of local search methods such as (API,API2,DM with all its neighbors (insert,swap,swap2,insert-swap), API2-swap, API2-swap2) and TTHM, we compare their values with the optimal solution (obtained by our (BAB) algorithm).Thirty-six test problems for n =10, 20, 30, 40 are tested and the results are given in table (3) . It is clear from table (3) that the best results as measured by the number of optimal solutions generated and the maximum deviation is obtained by (TTHM), followed by (API), (DM-swap), (DM-insert), (DM-in-sw), (DM-swap2), (API2-sw), and then (API2-sw2).

Table -3: comparative computation results

	n	10	20	30	40
API	NT	10	14	6	12
	MD	30	30	35	28
API2	NT	-	-	-	-
	MD	33	56	68	102
DM- insert	NT	8	9	8	-
	MD	36	39	38	37
DM – swap	NT	10	14	7	-
	MD	29	14	12	36
DM – swap2	NT	-	-	-	-
	MD	38	44	69	102
DM – in-sw	NT	3	6	3	-
	MD	35	32	34	36
API2 – sw	NT	1	-	-	-
	MD	31	37	78	98
	NT	-	-	-	-

API2- sw2
MD 38 52 88 102
NT 34 33 30 31
TTHM
MD 13 6 10 7
OPT NU - - - 8 AND 121.166 5321.888 111526.888 3653667.194

NT : Number of times out of 36 that an optimal solution is found or number of the best solutions for the unsolved problem.
MD : The maximum deviation.
NU : The number of unsolved problems.
AND: Average number of nodes.
OPT : Optimal solution by using BAB method.

Comparative Results for Local Search and Heuristic Method

Table (4) shows the comparative of local search methods that described in section (7) which are (API) and (DM-in-sw) with the heuristic (TTHM) that described in section (8). These methods are compared by listing for each value of n the number (NO) of times out of 36 that an optimal solution is found or the number the best solutions for the unsolved problems, and the maximum deviation (MD).

The results in table (4) show that (TTHM) has good performance and efficient results, followed by (API) and (DM-in-sw) methods which give reasonable results. It should be noted that in case of time, the local search methods (API), (DM-in-sw) is better than (TTHM).

Table -4: comparative computation results: 50≤ n ≤ 150 for local search methods

Method	**API**		**DM – in – sw**		**TTHM**	
n	NO	MD	NO	MD	NO	MD
50	12	32	-	68	30	7
75	7	39	-	107	34	6
100	4	48	-	145	34	2
150	5	34	-	254	34	5

Conclusions and Future Work

This study presents optimal and near optimal schedules for the problem of scheduling independent jobs on a single machine to minimize the total earliness penalties subject to no tardy jobs. This problem is considered to be NP-hard.

The study proposes a branch and bound algorithm (BAB) to solve our problem that employ a new lower bound scheme. This new lower bound is derive by using a relaxation on the earliness cost to be used in (BAB) to find optimal schedule, up to 40 jobs in reasonable time. Some special cases of our problem which leads to optimal solution are proved. Also, we report on the results of extensive computations tests of the following methods: Tree type heuristic method (TTHM), descent method (DM), adjacent pairwise interchange method (APIM) and a new method that linking descent method with adjacent pairwise interchange method. The main conclusion to be drawn from our computation results is that (TTHM) is more effective method for our problem, followed by (API) and (DM) methods. This study presents four heuristic methods for the problem.

An interesting future research topic would involve experimentation with same problem without the condition, subject to no tardy jobs.

REFERENCES

1. Supachai Pathumnakul , Pius J. Egbelu , Algorithm for minimizing weighted earliness penalty in single-machine problem .European Journal of Operational Research 161 780-796 (2005)
2. Alidaee, B., & Rosa, D. Scheduling parallel machines to minimize total weighted and unweighted tardiness . Computers and Operations Research , 24(8) , 775 – 788 (1997).
3. Du J. and Leung , Minimizing total tardiness on one processor is NP-hard , Mathematics of Operations Research 15 , 483-495 (1990).
4. Jorge M. S. Valente and Rui A. F. S. Alves , Improved lower bounds for the early /tardy scheduling problem with no idle time . Working paper , Faculdade de Economia do Porto , Portugal (2003).
5. Baker, K. R., And Scudder, G. D. Sequencing with earliness and tardiness penalties : A review . Operations Research 38 , 22-36 (1990).
6. LI, G. Single machine earliness and tardiness scheduling. European Journal of Operational Research 96 , 546-558(1997)
7. Graham R.L. , Lawler E.L. , Lenstra J.K. , Rinnooy kan A.H.G. " Optimization and approximation in deterministic sequencing and scheduling " . a survey . Ann Discrete Math , 287- 326(1979)
8. Chand S , Schneebergerh. Single machine scheduling to minimize weighted earliness subject to no tardy jobs .European Journal of Operational Research ; 34: 221-30(1988).
9. Abdul-Razaq T. S., " Machine scheduling problem, a branch and bound approach " Ph. Thesis, University of Keele,UK (1987).
10.Abdul-Razaq T. S. , and Potts C. N. , Dynamic programming state space relaxation for single machine scheduling , Journal of the

Operational Research Society 39, 141-152(1988) .

11.Hurink J. An exponential neighborhood for a one-machine batching problem , University of Twente , The Netherlands ,July (1998).