Flow- Shop scheduling Problem to Minimize Total Weighted Late Work

Al Zuwaini M. K.
Thi-qar University
Collage of Mathematics and Computer science

Abdul Razaq T. S.
University of Al Mustansiriyah
College of Science/Math. Dep.

Al Saidy S. K.

Abstract

In this paper we considered a flow-shop scheduling problem to minimize the total weighed late work. Branch and bound method was suggested to obtain an optimal solution. Several local search methods: descent method, simulated annealing, threshold accepting, genetic algorithm and hybrid method were used to obtain near optimal solutions, and comparison the results.

Keywords. Flow-shop, Late work, Schedule.

1 Introduction

Jacek et al. (2004) [10] provides the formal definition of the late work parameter, especially in the shop environment, together with its practical motivation. It contains general complexity studies and the results of investigating open-shop scheduling cases, i.e. two polynomial time algorithms for problems $O / pmtn, r_i / Y_w$ and $O_2 / d_i = d / Y$, as well as the binary NP- hardiness proof for $O_2 / d_i = d / Y_w$, where the late work $Y_i = min \{ D_i , P_i \}$,
$D_i = max \{0, C_i – d_i\}$ denotes the tardiness for task T_i.

The late work based criteria belong to the group of performance measures involving due dates. However, classical criteria, formulated for problems with due dates, such as e.g. the maximum lateness, total tardiness, calculate the penalty for solutions where some jobs exceed their due dates with respect to the time of their completion. In some applications, the penalty should be determined with reference to the amount of the late work independently of the completion time of a job. In the case of the late work criterion, only the amount of late work is important, if the whole job is delayed.

The late work criterion was first proposed in the context of parallel machines by Blazewiez (1984) [3], who showed the strong NP- hardness of the problem $P/ r_i /Y_w$. The proof concept is based on the complexity analysis for the minimal mean tardiness problem [10].The preemptive case $P/ pmtn, r_i /Y_w$ is polynomial solvable by a transformation to a min-

cost flow problem. It results in an algorithm of the overall complexity O (n^7 log n), where n denotes the number of tasks. This approach was further extended by Jacek et al. [10] to the case of a fixed number of uniform machines $Qk\ /pmtn,\ r_i\ /\ Y_w$. They proposed an $O\ (k^3\ n^7\ log\ kn)$ method, where k denotes the number of machines and n equals the number of tasks.

The late work criteria can be analyzed in agriculture, especially in all cases concerning perishable goods, as for example harvesting [13]. In this case, tasks represent different stretches of land that have to be harvested. Because they differ in climate and soil conditions as well as in the corn culture, they have differ times at which crops collecting should be started and finished. Processing times estimate quantities of crops. After a given due date, crops perish causing financial loss. Minimizing the total late work is equivalent to minimizing the amount of wasted crops. Summing up, the late work criteria apply to all those scheduling problems that concentrate on the amount of late work delayed after a given due date not on the duration of this delay.

We extend comparative work on local search algorithms for scheduling problems by presenting a computational study for the problem of minimizing the sum of weighted late works of jobs in a permutation flow shop. Thus, for each job i (i =1,…,n), its processing time p_{ij} on machine j (j=1,…, m) is given, a non–negative due date d_i and weight W_i are specified. The objective is to find a schedule S, defined by a permutation of the jobs, which yields a completion time C_i for each job i that minimizes the cost function $G(S) = \sum_{i=1}^{n} W_i v_i$ where $v_i = min\ \{\ T_i\ ,\ p_{i2}\ \}$, m =2 (number of machines), $T_i = max\ \{C_{i2} - d_i\ ,0\ \}$ (the tardiness of job i), and the completion times C_{ij} are:

$$C_{12} = C_{11} + p_{12} \text{ and } C_{i2} = max\ \{C_{i1}, C_{(i-1)2}\ \} + p_{i2}\ ,\ C_{i1} = \sum_{k=1}^{i} P_{k1}$$

Since the total weighted late work on one machine is strongly NP-hard [4], and $1//\Sigma\ W_i v_i$ is a special case of our problem $F_2\ //\ \Sigma\ W_i v_i$ then, we assume that our problem is strongly NP-hard.

Our computational comparison involves three neighborhood search algorithms; these are multi-start descent, simulated annealing and threshold accepting. These methods are compared with our branch and bound algorithm when n (number of jobs) less than or equal 30 and are compared with genetic algorithm and hybrid method (genetic threshold accepting) when n is large (when number of jobs grater than 30 jobs). (i.e. the solutions which are obtained from the five methods are compared with optimal solution which is obtained from branch and bound algorithm).

2 Formulation of the Problem:

Mathematically, the flow-shop scheduling problem (*FSP)* can be started as follows. Given a set $N = \{1, 2, \ldots, n\}$ of n simultaneously available jobs is to be processed on two machines where each job $i \in N$ requires processing on machine 1 first and then on machine 2. A job once started on a machine must be completed on that machine without interruption (i.e, no preemption is allowed). For each job i, let a_i and b_i be the processing times of job i, $i \in N$ at the first and second machine, respectively (i.e., $a_i=p_{i1}$, $b_i=p_{i2}$, i=1,2,…n), d_i be its due date and W_i be its weighted time for job i. The restriction that a machine can process only one job at a time, and requiring that once started processing on job must be uninterrupted until its completion, and we assumed that all jobs arrive at the same time, (typically at time 0).

In this paper we considered two performance measures, a late work criterion: total late work (Σv_i) and total weighted late work ($\Sigma W_i v_i$), our scheduling problems can be formulated as follows:

$$\left.\begin{array}{l} Min\ G(s) = \sum W_j v_j \\ subject\ to: \\ \left.\begin{array}{ll} v_j \le T_j, & j = 1, 2, \ldots, n \\ v_j \le b_j, & j = 1, 2, \ldots, n \\ T_j \ge 0, & j = 1, 2, \ldots, n \\ a_j, b_j > 0, & j = 1, 2, \ldots, n \\ w_j > 0, & j = 1, 2, \ldots, n \end{array}\right\} (1) \end{array}\right\} \ldots(A)$$

And for $F2 // \Sigma v_i$ problem

$$\left.\begin{array}{l} \mathrm{Min}\, G1(s) = \sum v_j \\ subject\ \mathrm{to}:\ (1) \end{array}\right\} \qquad (B)$$

where v_j is late work of job j to be minimized . A feasible solution of the *FSP* can be built from a permutation of n jobs on each of the two machines.

3 Special Cases

Since *FSP* to minimize total weighted late work is NP-hard, it is of interest to study special cases in an attempt to locate an optimal solution. And we try reducing the size of problem by finding a job which precedes or succeeds all other jobs in an optimal schedule. The following cases are considered.

1. The case of equal processing times and equal weights is considered for which $p_{ij} = P$ and $W_i = W$ for $i = 1, 2, \ldots, n$, $j = 1,2$, where P and W are the positive integers.

Theorem (1)

For the case of common processing time (i.e. $a_i = b_i = P$, $i = 1, 2, \ldots, n$)
if there is a schedule S such that for each job i, $i \in S$ $iP \le d_i$, then the schedule S is an optimal schedule for $F2/ p_{ij} = P / \Sigma v_i$ problem, $i = 1, 2, \ldots, n$ and $j = 1, 2$.

Proof

Since $iP \le d_i$ ($i = 1, 2, \ldots, n$), then $(i+1)P - d_i < P$ for each $i \in S$, but it is clear that $(i+1)P = C_{i2}$ be the completion time of job i on machine 2. Hence, $C_{i2} - d_i \le P$. We know that the late work of job i is given by $v_i = min\{T_i, b_i\}$ and $T_i = max\{C_{i2} - d_i, 0\}$, $b_i = P$ for $i = 1, 2, \ldots, n$. Then if $T_i = 0$, for each i the theorem is proved, otherwise if

$$T_i = C_{i2} - d_i > 0,$$

$$v_i = C_{i2} - d_i = (i+1)P - d_i \le P = b_i.$$

$\sum_{i=1}^{n} v_i = \sum_{i=1}^{n} \left((i+1)P - d_i\right) = \frac{1}{2}(n^2 + 3n)P - \sum_{i=1}^{n} d_i$ (constant). Then, S be an optimal schedule. □

Proposition (1)

If there is a schedule S such that for each $i \in S$ $iP > d_i$, then, this schedule is an optimal for $F2 / p_{ij} = P / \Sigma v_i$ problem.

Proof

Since $iP > d_i$, then $(i+1)P - d_i > P$ that is $C_{i2} - d_i > P$, hence $v_i = P$ and $\sum v_i = \sum_{i=1}^{n} P = nP$ (constant), then S be an optimal schedule. □

2. If there exists a schedule S and a job $i \in S$ such that $d_i \ge C_{max}$, then job i early and discarded.
3. If there exists a schedule S and a job $i \in S$ such that $a_i \ge d_i$, then job i is late and discarded .

The second case is clear. For the third case $a_i + b_i - d_i \ge b_i$ but $a_i + b_i$ is a small as possible value of completion time on machine 2 with respect to job i. Hence the late work of job i is equal to b_i at any position for job i in the schedule S.

Heuristic H

To get an upper bound we modify Jonson's Algorithm for the *F2 / / F_{max}* problem to get a heuristic that may give an optimal or near optimal solution for our problem (A), as follow:

Step (1): Set k=1, ℓ = n.

Step (2): Set current list of unscheduled jobs N = {J_1, … , J_n} .

Step (3): Set $P_i = a_i / W_i$; $q_i = b_i / W_i$.

Step (4): Let $S_i = \underset{i \in N}{Min} \{ P_i , q_i \}$.

Step (5): If $S_i = P_i$, $i \in N$ then
schedule J_i in k_{th} position of processing sequence
$N = N - \{ J_i \}$
$k = k + 1$
Go to step (7).

Step (6): If $S_i = q_i$, $i \in N$ then:
schedule J_i in ℓ th position of processing sequence.
$N = N - \{ J_i \}$, $\ell = \ell - 1$.

Step (7): If $N \neq \phi$, go to step (4). Otherwise stop.

The following example illustrate the total weighted late work of schedule S which is obtain from heuristic *H* (i.e. to get an upper bound UB = G(s) = $\sum_{i \in s} W_i v_i$

Example1. Consider 6-jobs, 2- machines permutation flow–shop, total weighted of late work problem, specified by the date in table (1)

Table 1 Data for example (1)

I	1	2	3	4	5	6
a_i	3	7	5	7	1	8
b_i	2	5	6	3	3	4
W_i	3	2	4	5	7	2
d_i	8	15	13	19	8	20

We observe that when applied heuristic (*H*), we obtain the sequence *S*: (5, 3, 2, 6, 1, 4) and the value of flow shop total weighted late work

UB = G(s) = $\sum_{i \in s} W_i \, v_i$ =35.

4 A Lower Bound

In this section we derive two lower bounds for $F_2 \,//\, \sum W_i v_i$ problem that can be used to reduce the size of the search tree generated by a branch and bound procedure.

4.1 Single Machine Bounds

First: To construct a lower bound (*LB)* on machine M_1 denoted by LB_{M1}, by using the relaxation techniques. The lower bound LB_{M1} is obtained as follows. Let b_i =0 for each job *i* and the resulting problem is $1 \,//\, \sum W_i v_i$ where $v_i = Min\ \{C_i - d_i, a_i\}$. It is well known that the resulting problem is still NP-hard and consider by Hariri et al. (1993), Pinedo (1995) and Bryan & Bahram (2002)[4]. Also when W_i =1 for each job *i*, then the problem $1 \,//\, \sum v_i$ is NP-hard, and is considered by Potts and Van Wassenhove (1991, 1992) [14].

To make use of the lower bound presented by Potts and Van Wassenhove, we relax the constraint of weighs by assuming that $w = Min\{W_i\}$ and the lower bound of the resulting problem is $LB_{M1} = wT_{max} \leq \Sigma W_i \, v_i$. This means we are solving the $1//T_{max}$ problem, which is solving by *EDD* rule (i. e., the job *i* is sequenced before job *j* if $d_i \leq d_j$ ($i, j = 1, \ldots, n$).

Second: To construct the second single machine bound on machine M_2 denoted by LB_{M2}. If we relax the capacity constraints of machine M_1, that is machine M_1 can process more than one job at a time. i.e. the processing time of a job *i* (a_i) on machine M_1 be as release date of job *i* with respect to machine M_2 (i.e. $r_i = a_i \ \forall i \in N$). The resulting problem ($1/ r_i / \Sigma W_i v_i$) is NP-hard which is considered by Al Zuwaini 2006 [2]. To obtain a lower bound LB_{M2} we will use a lower bound which is stated in [2] . Our lower bound $LB = \text{Max}\ \{LB_{M1}, LB_{M2}\}$.

Example 2 Consider again the example (1) presented previously. The sequence obtained by using *EDD* rule is (1, 5, 3, 2, 4, 6), where LB_{M1}=22. The second lower bound from the second relax is LB_{M2} = 12.

5 Branch and Bound (BAB) Algorithm

This section describes a branch and bound algorithm which may employ any of the lower bound, LB_{M1}, LB_{M2} schemes described above.

The better of the two lower bounds is used to provide an initial lower bound (LB), in a root node of search tree, (i.e. $LB = Max\{ LB_{M1} , LB_{M2} \}$. Also at the root node of search tree the heuristic method (*H*) is used to schedule the jobs to provide an initial upper bound *UB*. If $LB = UB$, then *UB* is an optimal value of total weighted late work flow-shop problem, and we don't need branching.

The branch and bound (*BAB*) algorithms use a forward sequencing branching which generates a search tree for which nodes at level ℓ correspond to initial partial sequences in which jobs are sequenced in the first ℓ positions. A newest active node search selects a node from which to branch. To eliminate nodes we use the dominance theorem of dynamic programming. This test use an adjacent job interchange to compare the sum weighted late work of a permutation flow-shop for the two jobs most recently added to the initial partial sequence with the corresponding sum when these two jobs are interchanged in position: if the former sum is larger than the latter, then the current node it eliminated, while if both sums are same, (as in [4]) some convention is used to decide whether the current node should be discarded.

For all nodes that remain after the dominance test is applied, we compute the lower bound *LB* by using the lower bounds LB_{M1} or LB_{M2}. If the lower bound for any node is greater than or equal to the previously generated upper bound UB, then that node is discarded. The *BAB* algorithm continues in a similar 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 (i.e. lower bounds of all nodes in the scheduling tree are greater than or equal to (UB), a feasible solution with this (UB) is an optimal solution.

6 Local Search Heuristics

Local search heuristics are a collective noun for rather classical iterative improvement techniques and more recently, developed metaheuristics [9]. The common base of local search methods is that they all move iteratively through the set of feasible solutions *S*. The walk is restricted by the restriction on the choice of the next solution. Neighborhoods define is dependence on the current solution which is a set of candidates to which the walk may continue. The different local search methods like e.g. iterative improvement, simulated annealing or tabu search differ mainly in the way of a candidate is chosen from the neighborhood. In a local search algorithm, we start with an initial solution *s* and generate a neighbor s' of *s*. In an iterative improvement algorithm,

only moves from s to $s' \in N(s)$ (the set of neighbors) are accepted which improve the value of the two machines flow-shop for total weighted late work (the objective function value).

The local search heuristics give no a-priori guarantees on solution quality or computational time; they got very popular in the last two decades and are very often used to tackle problems arising from practice [11]. The local search algorithms have proved to be useful for finding good solutions to various hard optimization problems [6]. In this section we consider several methods of local search, descent method, simulated annealing, threshold accepting, genetic algorithm and hybrid method.

6.1 Descent Method (DM)

In descent method, to obtain an initial solution (sequence), we shall use heuristic *H* which is presented in section (3), as a current sequence. Let *s* be denoted to the current solution, and G(s) be the cost of the objective function (total weighted late work for permutation flow-shop). In any stage two types of neighborhood structures in our computational experiments:

Insert neighborhood. Remove a job from one position in the sequence *s* and insert (shift) it in another position (either before or after the original position). This means that in a permutation $\sigma = (\sigma(1), \sigma(2), ..., \sigma(n))$, select an arbitrary job $\sigma(i)$ and shift it to a smaller position *j*, *j*<*i*, or to a larger position *k*, *k*>*i*. Thus, we have $|N(\sigma)| = (n-1)^2$. In the same applications this neighborhood is used in a specialized version, where only right or left shifts of an arbitrary job are allowed for the generation of neighbor. Thus if we consider again the example (1) presented previously we see that for example (5, 2, 3, 6, 1, 4) and (5, 3, 6, 1, 4, 2) each of them represent a sequence denoted by s_1, which is a neighbors of the current solution defined by *s* = (5, 3, 6, 2, 1, 4).

(ii) Swap neighborhood. Swap or interchange the jobs from any two positions (no necessary are adjacent) in the sequence *s*. Thus s_2= (5, 1, 6, 2, 3, 4) is a neighbored of the solution *s* = (5, 3, 6, 2, 1, 4).The new solution s' was found among these neighborhoods such that $G(s') = Min\{G(s_1), G(s_2)\}$. By acceptance test, test whether to accept the move from s to s'. If the move is accepted (i.e. $G(s') < G(s)$), then s' replaces *s* as the current solution; otherwise, *s* is retained as the current solution. This operation terminates when the best solution generated; otherwise, return to the neighbor generation step.

6.2 Simulated Annealing (SA)

Simulated annealing *SA* was originally proposed by Metroplis et al. (1953) was first applied to combinatorial optimization problems by Kirkpatrick et al. (1983) and by Cerny (1985) [12]. SA uses a probabilistic acceptance test. Starting with an initial sequence *s* as a current solution, another solution s' is drawn uniformly from the neighborhood of *s*. If $G(s)$ and $G(s')$ are the costs of solutions *s* and s' respectively, we accept s' as the new current solution if $\Delta \leq 0$, where $\Delta = G(s') - G(s)$. In the case of $\Delta > 0$, sequence s' is accepted as a new starting solution with probability $\exp(-\Delta / T)$. Where *T* is a parameter known as the temperature, which changes during the course of the algorithm. Typically, *T* is relatively large in the initial stages of the algorithm so that many increases in the objective function are accepted, and then *T* decreases until it is close to zero in the final stages. The rule that defines *T* is called a *cooling schedule* [6].

In our application of simulated annealing we used Jonson's algorithm to obtain an initial sequence (current solution) and swap neighborhood to generate s' a neighbor of *s*. As in [7] we used Lundy-Mess cooling scheme to generated a single iteration at each *k* temperatures $T_1, \ldots, T_k$ which are related by

$$T_{k+1} = T_k / (1 + \beta T_k)$$

where β is a constant, which is expressed in terms of *k*, T_1 and T_k by

$$\beta = (T_1 - T_k) / ((k\text{-}1) T_1 T_k)$$

based on initial experiments, we set $T_1 = 1000$ and $T_k = 30$.

Under the Lundy-Mess cooling scheme, the temperature decreases faster than in a geometric scheme, and consequently more iterations are performed at lower temperatures. The performance of geometric scheme is often more sensitive to the choice of initial temperature T_1 [7].

6.3 Threshold Acceptance (TA)

Threshold accepting is a similar method to simulated annealing. Whereas simulated annealing uses a probabilistic acceptance rule for solutions that cause deterioration in the objective function value, threshold accepting is deterministic. In threshold accepting, a move is accepted provided that it does not increase the objective function value by more than *t*, where t is a threshold value. The threshold value plays a similar role to the temperature in simulated annealing. Thus, the common practice is to select a decreasing sequence of threshold values, so that *t* is a relatively high value initially, and is fairly low during the final

iterations [5]. Celia et al. (1996) [7] obtained the threshold value t_k at k iteration in the generation and evaluation of one neighbor) by:

$$t_k = (t_1 - t_\ell)\,(1 - (k-1)/(\ell - 1))^s + v_\ell$$

where ℓ is the total number of iterations, t_1 and t_ℓ are the initial and final threshold values, and s defines the rate of decrease of the threshold values. We starting by heuristic H (see section 3) to obtain an initial sequence as a current solution s, insert neighborhood using at any stage to obtain a neighbor s' as a new solution. If $G(s)$ and $G(s')$ are denoted to costs of s and s', respectively, then the moves from s to s' are accepted if and only if $\Delta \le 0$ or $\Delta < t$, where $\Delta = G(s) - G(s')$. To set the threshold values t_k, for $1 \le k \le \ell$, we use a linear decreasing scheme as in [9].

$$t_k = t_{k-1} - (t_1 - t_\ell) / (\ell - 1)$$

Where t_1 and t_ℓ are the initial and final threshold values, respectively. These threshold values are computed from the objective function value of the initial solution $G(s)$ by setting $t_1 = 0.2\,G(s)$ and $t_\ell = 0.0001G(S)$.

7 Genetic Algorithm (GA)

Genetic algorithms are probabilistic search techniques based on the mechanism of evaluation. The solution space is usually represented by a population. In contract to other meta-heuristics such as simulated annealing and threshold accepting that operates with one solution, GA_s start with an initial population randomly generated or obtained by using any other heuristic. This population is composed by chromosomes, which are sets of genes. GAs tries to make progress in the population until a stopping condition is reached. The progressing from one population to another is obtained by applying two basic operators called crossover and mutation. At each generation, couples of chromosomes are selected from the current population, according to a specific criterion of selection. Then, the crossover operator that consists in combining these two selected chromosomes is applied to obtain two new chromosomes. After that, the mutation operator is applied in order to modify slightly the characteristics of same chromosomes. Two commonly used approaches for chromosome representation are indirect and direct encoding. In direct encoding, a chromosome completely represents a solution, while in indirect encoding a chromosome only gives parameter for generating a solution [15]. i.e. a chromosome is an n-length string representing a priority rule between jobs.

In our application we use a direct encoding with random keys to obtain a feasible sequence and evaluate its total weighted late work of the

two machines permutation flow-shop. The two points crossover in which two randomly selected crossover points are used. Select two positions at random as crossover points. For example, suppose that we start with two job sequences,

1-2-3-4-5-6-7

2-3-1-5-4-7-6

and we perform crossover at elements two and six, removing jobs 2 and 6 from the sequence one (*parent* 1) and replacing them in positions of jobs 3 and 7 respectively in sequence two (*parent* 2)and removing jobs 3 and 7 from sequence two and replacing them with jobs 2 and 6 respectively in sequence one, the resulting job sequences will be:

1-3-3-4-5-7-7

2-2-1-5-4-6-6

The two new job sequences are obviously not feasible solutions. Sequence one has jobs 3 and 7 listed twice and has omitted jobs 2 and 6. Sequence two has jobs 2 and 6 listed twice with jobs 3 and 7 omitted from the sequence. As in Beacn (1994) and [4], to avoid this possibility we used random keys within the vectors to act as surrogates for the final job sequence. Vectors of random numbers selected from 0 to1 are generated to represent the parents. Consider the following sequences of random numbers.

0.23 – 0.45 – 0.12 – 1.0 – 0.61 – 0.73 – 0.07

0.64 – 0.35 – 0.04 – 0.38 – 0.5 – 0.81 – 0.92

Now if we perform crossover at elements two and six, we obtain the following new vectors of random numbers.

0.23 – 0.35 – 0.12 – 1.0 – 0.61 – 0.81 – 0.07

0.64 – 0.45 – 0.04 – 0.38 – 0.5 – 0.73 – 0.92

To transform the offspring above into a feasible job sequence we start with the smallest number and place its position number in the first position of the job sequence, and so on until all elements are assigned. The random number vectors above would represent the following two sequences:

3 - 4 -2 -7- 5 - 6 -1

5 -3 -1- 2- 4 - 6 -7

The total weighted late work of permutation flow-shop evaluated for each parent and save the best cost (minimum cost) with its sequence. If it happened that at the crossover points the elements in parent one equal to

elements in parent two, then the swap neighborhood applied on the parent one before the crossover operation. The processing continue and its stopping after (1000) iteration.

8 Hybrid genetic threshold accepting (GTA) algorithm

Number of studies concludes that a standard genetic algorithm sometime performs poorly and that improvements may be achieved by incorporating neighborhood search. Celia et al. (1996) [7] used incorporating a descent algorithm into the genetic algorithm for the permutation flow shop to minimize the total weighted completion time. Allahverdi and Aldowaisan (2004) [1] proposed a hybrid simulated annealing and a hybrid genetic heuristic, which can be used for the single criterion of makespan or maximum lateness, or the bicriteria problem. Goncalves et al. (to appear) [8] proposed a hybrid genetic algorithm for the job shop scheduling problem. After a schedule is obtained using the priorities defined by the genetic algorithm, a local search heuristic is applied to improve the solution. We therefore adopt the following hybridization: A genetic threshold accepting (*GTA*) algorithm was developed by incorporating the threshold accepting algorithm which described in section (6) into the genetic algorithm of section (7). Threshold accepting was applied first to each current solution to obtain best parents so that each new parent is a local optimum for our objective function, and then applied a crossover and mutation. The structure of our *GTA* algorithm is the same as that of genetic algorithm of section (7) except for the threshold accepted used to reduce the population and improvements the parents. Nevertheless, to be extent of our knowledge, no hybrid metaheuristics was proposed for problem considered in this chapter.

9 Computational Experience

The set of problems tested with ten different numbers of jobs (n=10, 20, 30, 35, 50, 100, 500, 1000, 5000 and 10000) are generated as follows: For each value of n four integers processing times a_i, b_i, weighted time W_i and due date d_i (i=1, 2, …, n), a_i, b_i and W_i were generated from the uniform distribution $[\alpha_1 T, \alpha_2 T]$ where $T = \sum_{i=1}^{n} p_i$ and $\alpha_1 \in \{0.2, 0.4, 0.6, 0.8\}, \alpha_2 \in \{0.4, 0.6, 0.8, 1.0\}$, $\alpha_1 < \alpha_2$. Due date d_i for every problem were generated as in [2] (section 6.11). For each select value of n, two problems were generated for each of ten pairs of α_1, α_2 producing 20 test problems for each value of n. We propose a branch and bound algorithm to solve problem (A) with up to 30 jobs. Different local search

metaheuristics, for some methods (descent *DM*, simulated annealing *SA*, threshold accepting *TA*, genetic algorithm *GA* and hybrid method *GTA*).The solutions value found by the local search methods and the optimal value of *BAB* algorithm are compared in table (2). Table (3) gives the deviation of the average of each local search heuristics from the best solution for all local search methods. Figure (1) illustrates average CPU (seconds) required by each method

Table (1): Comparative the results of the local search with the optimal solution (n=30) for the problem $F2//\sum W_i v_i$

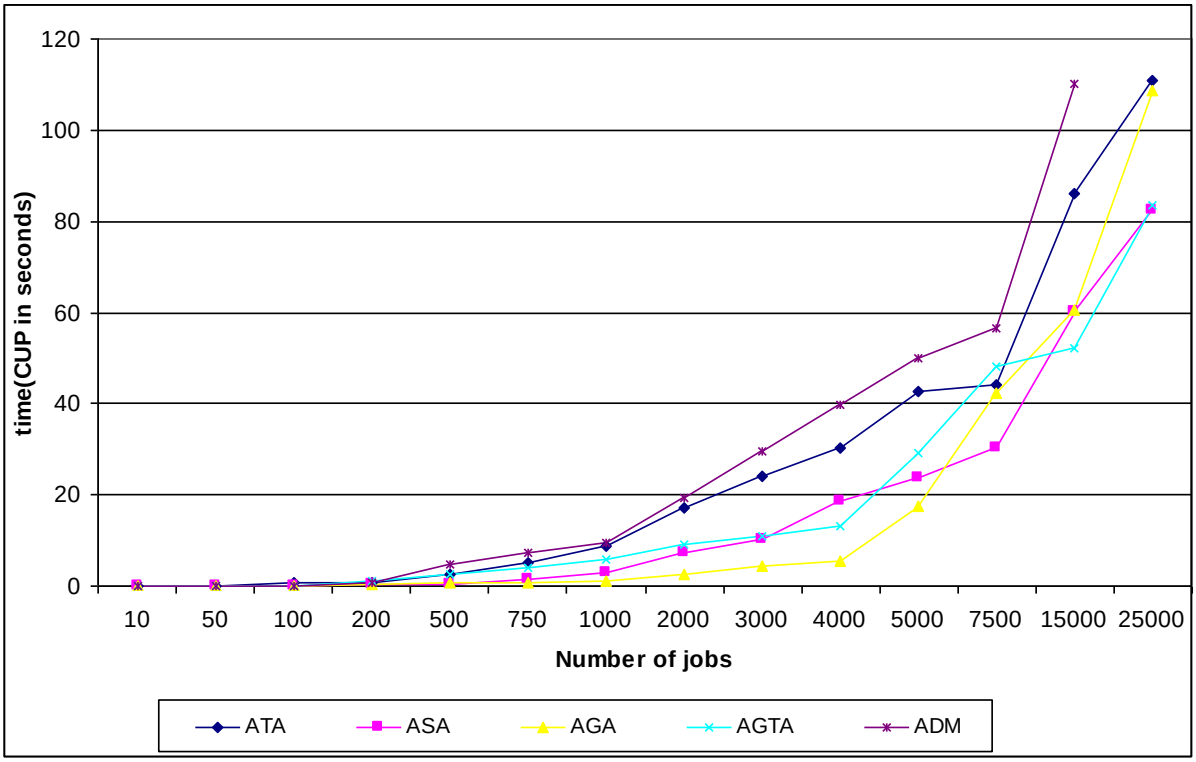
No.	DM	time	GTA	time	GA	time	TA	time	SA	time	BAB	time
1	486	.006	472	.002	534	0	472	0	472	0	471	45
2	518	.006	480	0	549	0	483	0	480	0	477	43
3	619	.011	592	.06	630	0	592	0	598	0	491	50.2
4	816	.011	816	0	852	0	816	0	816	.05	808	44
5	892	.11	905	.05	910	0	905	0	464*	0	464	44
6	561	.17	531	0	514	0	533	0	552	.06	508	50.2
7	1288	.17	1210*	.06	1238	0	1278	0	1278	0	1210	36.5
8	906	.22	875	.05	880	.06	924	0	909	0	872	50
9	435	.22	440	0	448	0	450	0	429	0	423	33
10	448	.28	420*	.06	431	0	438	0	447	0	420	28
11	467	.22	426*	0	430	0	447	0	457	.06	426	28
12	551	.33	551	.05	561	0	551	0	543	0	525	33
13	466	.17	438*	0	447	0	448	0	445	.05	438	28
14	475	.39	456	0	478	0	456	0	456	0	452	28
15	671	.39	636	.05	634	.05	646	0	646	.06	625	44
16	613	.44	606*	0	614	0	613	0	613	0	606	50.2
17	951	.33	945	.06	1000	0	945	0	887	0	854	43
18	923	.44	890	0	897	0	923	0	915	.05	885	43
19	908	.50	908	.05	936	0	908	0	886	0	868	43
20	470	.50	469	0	503	0	469	0	466	.06	466	44

*: Indicates that the problem has an optimal solution equals to the heuristic value.

Table (2): Averages deviations about the best average for the problem $F2//\sum W_i v_i$

n.	deviations of averages				
	DM	TA	GA	SA	GTA
50	163	0	58	0	0
100	87	0	42	4	0
200	9130	121	84	0	21
500	1431	242	188	3	0
750	1575	4121	149	0	460
1000	8392	62	45	0	38
2000	0	190	813	192	90
3000	1504	0	960	1042	0
4000	40316	0	2647	15150	0
5000	43066	6681	6114	13615	0
7500	199630	4790	22870	4840	0
15000	1529	1151	1680	146080	0
25000	-	83300	115970	639600	0

Fig.(1): Time averages for the problem $F2//\sum W_i v_i$



10 Conclusion and Future Work

In this paper we considered a flow-shop scheduling problem to minimize the total weighted late work; we assume that job $i (i \in N)$ becomes available for processing at time zero. We suggested two lower bounds, and propose branch and bound algorithm to solve the flow-shop scheduling problem for small instances. Practical experiments show that only instances with up to two machines and 30 jobs can be solved optimally.

To solve large size problems, we propose five local search heuristic. We evaluate its performances by comparison with an optimal value for small size and with a best value from all five methods for large size. However, we do not considered the case of transportation times of the late work, from one machine to another. To be extent of our knowledge no studies consider this case, and we shall let the problem $F2/\ell_i/\sum W_i v_i$, two machines flow-shop with transportation time between the machines, to minimize the late work as a future work.

References:

[1] Allahverdi, A. and Aldowaisan, T. "No-Wait Flow Shops with Bicriteria of Makespan and Maximum Lateness", *European Journal of Operational Research* 152 (2004) 132-147.

[2] Al-Zuwaini M. K. "A Comparative Study of Local Search Methods for Some Machine Scheduling Problems With Usage of Hybridization As A Tool", Ph. D. Thesis, Math. Dep. College of Education, Ibn Al Haitham, Baghdad University, April, 2006

[3] Blazewies, J. "Scheduling Preemptible Tasks on Parallel Processors swith Information Loss", *Research Technique ET Science Informatiques* 3 (1984) 415-420.

[4] Bryan, R. k. and Bahran Alidaee, "Single Machine Scheduling to Minimize Total Weighted Late Work: a Comparison of Scheduling Rules and Search Algorithm", *Computers and Industrial Engineering* 43 (2002)509-528.

[5] Crauwels, H. A. J. "Local search heuristics for single machine scheduling with batch set-up times to minimize total weighted completion time", *Annals of Operations Research* 70 (1997) 261-279.

[6] Ebbe G. Negenman, "Local Search Algorithms for the Multiprocessor Flow Shop Scheduling Problem", *European Journal of Operational Research 128 (2001)* 147-158

[7] Celia A. Glass and Chris N. Potts, "A Comparison of local search methods for flow shop scheduling", Annals operations Research 63 (1996) 489-509.

[8] Goncalves, J. F., Mendes, J. J. M. and Resende, M. G. C., "A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem", *European Journal of Operational research,* To Appear

[9] Herman Grauwels, "A Comparative Study of Local search Methods for One-Machine Sequencing Problems*", Ph. D. Thesis, Katholieke Universities Leuven Faculteit Toegepaste Wetenschappen Department Werktuigkunde Centrum voor Industried Beleid Celestijnenlaan 300A-300L Leuven,* Mei 1998.

[10] Jacek Blazewiez, Erwin Pesch, Malgorzata Sterna and Frank Werner, "Open Shop Scheduling Problems with Late work Criteria", *Discrete Applied Mathematics* V.134. Issue 1.3 (January 2004), 1-24.

[11] Jahaun Hurink, "Solving Complex Optimization problem by Local Search", *Fachbereich Mathematikl Informatik der Universitat Osnabriick* (*1998*).

[12] Jatinder N. D. Gupta, Karsten Hennig, Frank Werner, "Local Search Heuristics for Two-Stage Flow Shop Problems with Secondary Criterion" *Computers and Operations Research* 29(2002) 123-149.

[13] Potts, C. N. and L. N. Van Wassenhove, "Single Machine Scheduling to Minimize Total Late Work", *Operations Research* 40-3(1991)586-595

[14] Potts, C. N. and L. N. Van Wassenhove, “Approximation Algorithms for Scheduling a Single Machine to Minimize Total Late Work”, *Operations Research Letters* 11(1992)261-266.

[15] Riad Aggoune “Minimizing the Makespan for the Flow Shop Scheduling Problem with availability Constraints”, *European Journal of Operational Research* 153 (2004) 534-543.