Journal of Kerbala University , Vol. 10 No.3 Scientific . 2012

Scheduling jobs with release dates on identical machines to minimize weighted completion times function

Hanan Ali Chachan
Mathematical Department / College of Science
University of Mustansiriya / Baghdad; Iraq.

Abstract:-

This paper considers the problem of scheduling independent jobs with release dates on m identical machines to find minimize the total weighted completion times. The purpose of this paper is to describe meta-heuristic algorithms such as Memetic algorithm approach (MA),Threshold acceptance algorithm (TA) and Tabu search (TS), in order to find near optimal solution (feasible solution) to minimize the total weighted completion time which subject to release dates. The problem denoted as $P|r_j|\sum_j w_j C_j$.

المستخلص:

في هذا البحث تناولنا مسألة جدولة الأعمال بمواعيدِ العرض على m من المكائن المتماثلة لإيجاد تَقليل أوقاتِ الإكمالِالمرجّحةِ الكليّةِ بشرط وجود وقت تحضير للأعمال. نُطبّقُ بَعْض طرقِ البحثِ المحليّةِ مثل نظرةِ خوارزميةِ (MA)، خوارزمية (TA) و (TS) لإيجاد حلول مثالية مقبولة للمسألة $P|r_j|\sum_j w_j C_j$.

Key words: Machine scheduling, parallel machines, Meta-heuristic, Tabu search, Memetic algorithm, Threshold acceptance algorithm.

1. Introduction:-

A machine scheduling problem is an extended field of research in various applications. The main elements of machine scheduling problems are machine configuration job characteristics, and objective function. The machine can be classified to single and multiple machine problems in a broad sense. Parallel machine scheduling problems can be referred as a class of problems that relaxed from the multiple machine scheduling problems [3]. In an identical parallel machine system all machines are identical and job can be processed by any free machine. Many researchers studied parallel machine scheduling problems in past. Cheng and Sin [3] surveyed a parallel machine scheduling problem. Brone et.al. [2] proved that even a two-machine system for finding the weighted sum of flow times with an unequally weighted set of jobs is NP-hardness. Ramachandra and Elmaghraby [8] proposed a Binary Integer Program (BIP) and a Dynamic Program (DP) on two machines to minimize to weighted completion time. Nessach et.al. [10] presented an identical parallel machine scheduling problem with release dates to minimize the total completion time. They proposed heuristic algorithm based on this condition to build a schedule belonging to subset and they developed the lower bound computed in polynomial time. Leung et.al. [7] analyzed efficient heuristic for the scheduling orders for multiple product types to minimize the total weighted completion time without release dates.

The purpose of this paper is to describe meta-heuristic algorithms in order to find near optimal solution (feasible solution) to minimize the total weighted completion time which subject to release date. The problem denoted as $P|r_j|\sum_j w_j C_j$.

The rest of this paper is given blew. In section (2) details of the given problem. Sections (3,4,5,6) presents a description the local search methods. Computational results obtained by the proposed local search methods in section (7). In section (8) future research in this area.

2. Problem Description and Mathematical Formulation

In the parallel machines scheduling problem, a set of n independent jobs should be scheduled on m identical machines without preemption. Each job has a processing time p_j, a release date r_j and a due date d_j. All these jobs data are generated randomly. Some assumption must be respected: each machine can execute only one job at once, each job can be processed only once. Some notations are defined below:

n: the number of jobs
m: the number of machines
j: the index of jobs $j = 1, 2, \ldots, n$
k: the index of machines $k = 1, 2, \ldots, m$
r: the order of job in the machine.
r_j : the release date of job $j = 1, 2, \ldots, n$
d_j : the due date of job $j, j = 1, 2, \ldots, n$
p_j : the processing time of job $j, j = 1, 2, \ldots, n$
C_j : the completion time of job $j, j = 1, 2, \ldots, n$
n_k : the number of jobs assigned to machine *k*.

The problem can be formulated as follows:

$$minimize \sum_j w_j C_j \quad \ldots (1)$$

Subject to

$$\sum_{j=1}^{n} x_{jkr} = 1,\ k = 1, 2, \ldots, m,\ r = 1, 2, \ldots, n_k \quad \ldots (2)$$

$$\sum_{k=1}^{m}\sum_{j=1}^{n_k} x_{jkr} = 1,\ j = 1, 2, \ldots, n \quad \ldots (3)$$

$$p_{[kr]} = \sum_{j=1}^{n} x_{jkr}\ p_j\ ,\ k = 1, 2, \ldots, m,\ r = 1, 2, \ldots, n_k \quad \ldots (4)$$

$$r_{[kr]} = \sum_{j=1}^{n} x_{jkr}\ r_j\ ,\ k = 1, 2, \ldots, m,\ r = 1, 2, \ldots, n_k \quad \ldots (5)$$

$$d_{[kr]} = \sum_{j=1}^{n} x_{jkr}\ d_j\ ,\ k = 1, 2, \ldots, m,\ r = 1, 2, \ldots, n_k \quad \ldots (6)$$

$$C_{[kr]} = \max\left(C_{[kr-1]}, r_{[kr]}\right) + p_{[kr]},\ k = 1, 2, \ldots, m,\ r = 1, 2, \ldots, n_k \quad \ldots (7)$$

$$x_{jkr} = 0\ or\ 1,\ k = 1, 2, \ldots, m,\ r = 1, 2, \ldots, n_k,\ j = 1, 2, \ldots, n_k \quad \ldots (8)$$

$$y_{ij} = 0\ or\ 1,\ i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, m \quad \ldots (9)$$

$$w_i \geq 0,\ i = 1, 2, \ldots, n \quad \ldots (10)$$

Equation (1) represents the objective function and the goal of our work is to minimize the total of weighted completion time. Constraint (2) ensures that only one job can be scheduled at the *r-th* job position. Constraint (3) means that each job can be scheduled only once. Constraints (4)-(7) denote the data of jobs which are scheduled at the *r-th* job position of *k-th* machine position jobs, such as processing times, release dates, due date, and completion time. Constraint (8) is a decision variable, if job *j* is scheduled on machine *i* in position *r*, then $x_{jkr} = 1$, otherwise 0. Constraint (9) shows that if job *j* is the immediate successor of the job *i* on the some machine, then $y_{ij} = 1$, otherwise, $y_{ij} = 0$.Weighted for each job *i*, $i = 1, 2, \ldots, n$ in condition (10).

3. Local search heuristics:-

Research a local search in scheduling is quite extensive, but applications to parallel machine scheduling are scarce. There are few computational studies that compare different local search methods on the same scheduling problem [1]. Three local search algorithms are implemented in sections (4,5,6).

First we introduce some neighborhoods types which are used in local search methods

3.1 Neighborhood generating Mechanisms:-

We develop a local search methods here where five operations are used to generate local search neighborhoods, these operations are the,

- **Move operation:-**

Reassigning one job from a machine with minimum total weighted completion time to another machine. For instance if we have 10 jobs and M1 and M2 denoted to machines, then:

M1: 2 **3** 9 4 5   M1: 2 9 4 5
M2: 10 1 6 7 8   ⇒   M2: 10 **3** 1 6 7 8

- **Swap operation:-**

Swap one job from a machine with minimum total weighted completion time with one job from another machine. For instance:

M1: 2 3 **9** 4 5   M1: 2 3 **1** 4 5
M2: 10 **1** 6 7 8   ⇒   M2: 10 **9** 6 7 8

- **Insert $[i, j]$ operation:-**

Represent a move where job *i* is remove from machine *m*(*j*) such that [let *m*(*j*) denote the machines that job *i* is currently processed on] and inserted right before. For instance:

M1: 2 3 9 **4** 5   M1: 2 3 1 5
M2: 10 1 6 7 8   ⇒   M2: 10 9 6 7 **4** 8

- **Insert $[j, p(l)]$ operation:-**

Denote the move where job *j* is scheduled to be processed at the end of machine *l*. For instance:

M1: 2 3 9 **4** 5   M1: 2 3 1 5
M2: 10 1 6 7 8   ⇒   M2: 10 9 6 7 8 **4**

- ***k*-insert operation:-**

We construct a restricted version of the k-insert neighborhood by only allowing moves insert $[i_1, j_1]$ insert $[i_2, j_2]$ …, insert $[i_k, j_k]$ where $i_l < i_{l+1}$ for *l* = 1 to *k* – 1 with $j_l < j_{l+1}$ for *l* = 1 to *k* – 1. For instance:

M1: 2 3 9 4 5   M1: 9 4 5
M2: 10 1 6 7 8   ⇒   M2: 10 **2** 9 6 **3** 7 8

Now we introduce algorithm (1) which is generated feasible solution we can improved it by applied at initial solution (*ini*) which describe at below.

Algorithm(1):-

Swaps as kick moves for parallel machines scheduling.

Procedure kick move (s)
For M time do
Randomly select two machines *k* and $l \ni k \neq l$
Randomly select two jobs $m_k(i)$ and $m_l(j)$
apply swap $[m_k(i), m_l(j)]$
End for

Where M dependent on the number of machines *m*. In our experiments, we discovered that choosing M randomly from interval (0.3m, 0.8m).

After algorithm (1) the jobs assigned to each machine are ordered by NEH algorithm [10].

Initial solution (*ini*):-

$J_i^k(l)$ denotes job j_i which is placed in the *l-th* position on machine *k*. The initial solution is generated as follows:

Step(1):

Arrange all jobs by SRT (shorted release dates). And obtain a sequence $\{J_i(f), f = 1,2,...,n\}, J_i(f)$ means that job J_i is placed in the *f-th* position on the SRT sequence.

Step(2):-

$k \leftarrow 1, f \leftarrow 1, l \leftarrow 1$

Step(3):-

$J_i^k(l) \leftarrow J_i(f)$

Step(4):-

$k \leftarrow k+1, f \leftarrow f+1$

Step(5):-

If $k > m$, then $k \leftarrow 1$, and $l \leftarrow l+1$ if $f > n$ stop otherwise go to step(3).

l^k denote the number of jobs assigned to machine *k*.

Now, we give details about local search methods which are used to solve $P|r_j|\sum_j w_j C_j$ problem.

4. Memetic Algorithm Approach

Memetic algorithms (MAs) (Moscato, 1989), combines the recognized strength of the population-based methods with the intensification capability of a local search. In an MA, all individuals of the population evolve solutions until they become a local minima of a certain neighborhood (or highly evolved solutions of individual search strategies), i.e., after the recombination and mutation steps, a local search is applied to the resulting solutions. A more formal introduction to MAs and polynomial merger algorithms can be found in Moscato (1999). Figure 1 shows a pseudo-code representation of a local search-based memetic algorithm.

```
1.  procedure Local Search-based Memetic Algorithm;
        BEGIN
2.          Initialize Population Pop using First Pop();
3.          For Each individual i ∈ Pop DO i:= Local-Search(i);
4.          For Each individual i ∈ main Pop DO Evaluate Fitness(i);
            REPEAT /*generation loop */
5.              FOR i:= 1 to #recombinations DO
6.                  Select To Merge a set S_par ⊆ Pop;
7.                  offspring:=Recombine(S_par, x);
8.                  IF (select To Mutate offspring) THEN offspring := Mutate (offspring);
9.                  offspring:= Local-Search(offspring);
10.                 Evaluate Fitness(offspring);
11.                 Add In Population individual offspring to Pop;
12.             End For;
13.             IF (Pop has_converged) Pop:= RestartPop(Pop);
            UNTIL stop criterion;
    END
```

Figure1. Pseudo-code of a memetic algorithm

The initialization part begins at **initialize Population** and ends just before the **repeat** command. This part is responsible for the generation, optimization and evaluation of the initial population (*Pop*). The second part includes the so-called 'generation loop'. At each step, two parent configurations are selected for recombination and an offspring is produced and, if selected to mutate, it suffers a mutation process. The next steps are local search, evaluation and insertion of the new solution into the population. If the population is considered to have lost diversity, a mutation process is applied on all individuals except the best one. Finally, a termination condition is checked.

4.1 Population Structure

In our implementation we use a hierarchically structured population organized as a complete ternary tree of individuals clustered in 4 subpopulations or clusters, as shown in figure 2. In contrast with a non-structured population it restricts crossover possibilities. Other studies have shown that the use of structured populations is more effective when compared to non-structured populations (e.g. França *et al.* 1999; Buriol *et al*. 1999).

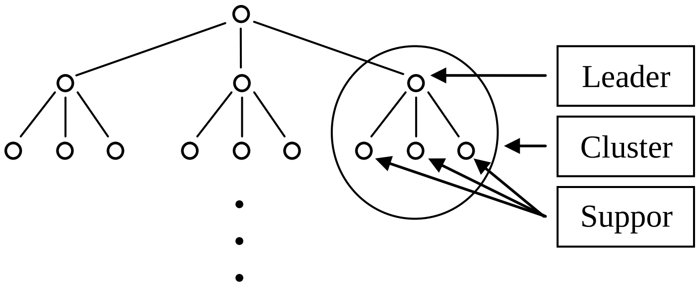

Figure 2. Population structure

The structure consists of several clusters, each one composed of a leader and three supporter solutions. The leader of a cluster is always better fitted than its supporters. This hierarchy ensures top clusters have better fitted individuals than bottom clusters. As new individuals are constantly generated, replacing old ones, periodic adjustments to keep this structure well-ordered are necessary. The number of individuals in the population is restricted to the numbers of nodes in a complete ternary tree: 13, 40, 121, etc. That is, 13 individuals are necessary to construct a ternary tree with 3 levels, 40 to one with 4 levels and so on.

4.2 Representation of Individuals

The representation we have chosen for the $P|r_j|\sum_j w_j C_j$ is quite intuitive, with a solution represented as a chromosome with the alleles assuming different integer values in the [1, *n*] interval, where *n* is the number of jobs. There are *m*-1cut-points in the chromosome that define the subsequences assigned on machine. For instance, < 4 9 6 * 2 8 5 1 * 3 10 7 > is a possible solution for a problem with 10 jobs. The cut-points (*) are in positions 4 and 9. Therefore, subsequence 1 executes operations 4 - 9 - 6, in this order; subsequence 2 executes operations 2 - 8 - 5 - 1 and subsequence 3 performs operations 3 - 10 - 7.

4.3 Recombination

The command **selectToMerge** indicates the task of selecting a subset of individuals (called $S_{par} \subseteq Pop$) to be used as input for the crossover operation, represented by the Recombine( ) function. In the pseudocode, the symbol '*x*' stands for the instance of the problem. In this case, since we are addressing the $P|r_j|\sum_j w_j C_j$, the '*x*' refers to matrix s_{ij} and vector p_j. The crossover operator implemented is the well-known Order Crossover (OX). After choosing two parents, a fragment of the chromosome from one of them is randomly selected and copied into the offspring. In the second phase, the offspring's empty positions are sequentially filled according to the chromosome of the other parent.

Parent A	2 4 * 7 6 3 * 1 5
Parent B	6 5 2 * 7 1 4 * 3
Initial Offspring	_ _ _ 7 6 3 * _ _ (A)
Construction phase	5 _ _ 7 6 3 * _ _ (B)
	5 2 _ 7 6 3 * _ _ (B)
	5 2 * 7 6 3 * _ _ (B)
	5 2 * 7 6 3 * 1 _ (B)
Final Offspring	5 2 * 7 6 3 * 1 4 (B)

In the example above, the fragment is selected from the parent A and consists of the alleles < 7 6 3 * >. The child's empty positions were then filled according to the order that the alleles appear in the chromosome of parent B. The number of new individuals generated in every iteration is controlled by a parameter named *cross_rate* which is expressed as the percentage of new individuals over the total population.

4.4 Mutation

In our method, a traditional mutation strategy based on job swapping was implemented. According to it, two positions are randomly selected and the alleles in these positions swap their values. The alleles that are swapped can be both related to two jobs (two integers) or one to a job and other to a cut-point. In the first case the number of jobs on each machine remains the same. In the second case the structure of the solution is changed, because the number of jobs on each machine is modified. The case in which both positions selected are cut-points does not change anything at all.

We implemented two mutation procedures - Mutate( ) and RestartPop( ); the first can be considered a light mutation and the other is a heavy mutation procedure. The Mutate( ) function is applied to each individual with a probability of *mut_rate*and, once applied, it mutates two alleles. Implementations with more changes per individual showed no improvement. In fact, when the number of alleles to be mutated increases, valuable information tends to be lost, worsening the MA's overall performance. The RestartPop( ) procedure, on the other hand, mutates all individuals in the *mainPop* except the incumbent solution. The swapping procedure is applied to each individual 10*n* times, so the resulting population almost resembles a randomized restarting procedure.

4.5 Fitness Function

As in this problem the goal is to minimize the total weighted completion time which subject to release date, the fitness function was chosen as randomly.

4.6 Selection of Parents

Recombination is only allowed between a leader and one of its supporters and both are randomly selected. An intensification procedure was implemented, forcing the best individual to take part in approximately 10% of the crossovers. This procedure showed itself to be very effective when compared to a standard selection policy. Tests revealed small but repeated improvements over the scheme without intensification.

4.7 Offspring Insertion into Population

Once the leader and one supporter are selected, the recombination, mutation and local search take place and an offspring is generated. If the fitness of the offspring is better than the supporter's that took part in the recombination, the offspring replaces the supporter. If the new individual is already present in the population, it is not inserted in it. We adopted a policy of not allowing duplicated individuals to reduce loss of diversity. After the generation is over and all individuals were inserted, the population is restructured. The hierarchy forces the fitness of an individual to be lower than the fitness of the individual just above it in the ternary tree. Following this policy, the higher clusters will have leaders with better fitness than the lower clusters and the best solution will be the leader of the root cluster. The adjustment is made by comparing each individual to the individual just above which it is connected to. If the lower individual becomes better than the upper one, they swap places.

5. Threshold Acceptance Method (TH)

A variant of simulated annealing is the **threshold acceptance method** (Brucker 2007). It differs from simulated annealing only by the acceptance rule for the randomly generated solution $s' \in N$. s' is accepted if the difference $Z(s') - Z(s)$ is smaller than some non-negative threshold *t*. *t* is a positive control parameter which is gradually reduced.

Algorithm Threshold Acceptance

1. $i := 0$;
2. Choose an initial solution $s \in S$;
3. *best* :=$Z(s)$;
4. $s^* := s$;
   REPEAT /*generation loop */
5. Generate randomly a solution $s' \in N(s)$;
6. IF $Z(s') - Z(s) < t_i$ THEN $s := s'$;
7. IF $Z(s') <$ *best* THEN
   BEGIN
8. $s^* := s'$;
9. $best := Z(s')$;
   END;
10. $t_i + 1 := g(t_i)$;
11. $i := i + 1$
    UNTIL stop criterion;
    END

g is a non-negative function with $g(t) < t$ for all *t*.

Figure 3.Threshold acceptance structure

The threshold acceptance method has the advantage that they can leave a local minimum. They have the disadvantage that it is possible to get back to solutions already visited. Therefore oscillation around local minima is possible and this may lead to a situation where much computational time is spent on a small part of the solution set.

6. Tabu Search (TS)

In this section we describe the tabu search procedure used to solve the $P|r_j|\sum_j w_j C_j$ problem. Tabu search (see Glover and Laguna [1997] and Gendreau [2003]) is one of the most popular techniques to find near optimal solutions to hard combinatorial optimization problems. A simple way to avoid such problems is to store all visited solutions in a list called tabu list T and to only accept solutions which are not contained in the list. However, storing all visited solutions in a tabu list and testing if a candidate solution belongs to the list is generally too consuming, both in terms of memory and computational time.

To make the approach practical, we store attributes which define a set of solutions. The definition of the attributes is done in such a way that for each solution visited recently, the tabu list contains a corresponding attribute. All moves to solutions characterized by these attributes are forbidden (tabu). In this way cycles smaller than a certain length *t*, where *t* usually grows with the length of the tabu list, will not occur.

Besides a tabu status, a so-called aspiration criterion is associated with each attribute. If a current move leading to a solution s' is tabu, then this move will be considered admissible if s' satisfies the aspiration criterion associated with the attribute of s'. For example, we may associate with each attribute a threshold *k* for the objective function and allow a move *m* to a solution s' if $Z(s') \le k$, even though *m* is tabu.

The following algorithm describes the general framework of tabu search.

Algorithm Tabu Search

1. Choose an initial solution $s \in S$;
2. *best*:= $Z(s)$;
3. $s^* := s$;
4. Tabu-list:= φ;
   REPEAT /*generation loop */

5. *Cand*(*s*):= { $s' \in N(s)$ | the move from s to s' is not tabu OR s' satisfies the aspiration criterion};
6. Generate a solution $\bar{s} \in$ *Cand*(*s*);
7. Update the tabu list;
8. $s := \bar{s}$;
9. IF $Z(s)$ < *best* THEN
   BEGIN
10. $s^* := s$;
11. *best* := Z(*s*);
   UNTIL stop criterion;
   END

Figure 4.Tabu search structure

Different stopping criteria and procedures for updating the tabu list *T* can be developed. We also have the freedom to choose a method for generating a solution $\bar{s} \in$ *Cand*(*s*). A simple strategy is to choose the best possible $\bar{s}$ with respect to function *Z*:

$$Z(\bar{s}) = \min\{Z(s') \mid s' \in Cand(s)\} \qquad \ldots (1)$$

However, this simple strategy can be much too time-consuming, since the cardinality of the set *Cand*(*s*) may be very large. For these reasons we may restrict our choice to a subset $V \subseteq$ *Cand*(*s*):

$$Z(\bar{s}) = \min\{Z(s') \mid s' \in V\} \qquad \ldots (2)$$

Usually the discrete optimization problem (1) or (2) is solved heuristically.

7. Computational experience

7.1 Test Problems

In this section a number of experiments are carried out which outlines the effectiveness of local search algorithms described above. The purpose of these experiments is to compare the performance Memetic algorithm approach (MA), Threshold acceptance algorithm (TA) and Tabu search (TS) for $P|r_j|\sum_j w_j C_j$ Problem. These methods are coded in Matlab language R2009b and runs on a Pentium IV at 2.00GHz, 2.92GB computer. The job data include the processing times p_j, the due dates d_j and release dates r_j, For each job ,the processing times p_j and r_j are generated using a uniform distribution [1,100]. The due dates of jobs are in the interval [P*(1-TF-(RDD/2)), P*(1-TF+(RDD/2))] where $P = \sum_j p_j / m$ and we have chosen TF and RDD between 0.2 and 0.4. For the comparison of three local search method, three types of instances have been processed: the problem with (10, 20, 30, 40, 50, 75, 100, 150, 200, 500, 1000) jobs and (2,3,5) machines. The full enumeration method cannot be applied on large problems because of the too long execution times

7.2 Comparative Results

In this section we will report on the results of our computational tests to show the effectiveness of our local search methods. In Table(1) we compare the neighborhoods types (move, swaps, Insert [*i*, *j*] , Insert [*j*, *p*(*l*)]) with the problems (10, 20, 30, 40, 50, 75, 100, 150, 200, 500, 1000) jobs on (2,3,5) machines .

In the following tables (2,3) show the efficiency local search methods Memetic algorithm approach (MA), Threshold acceptance algorithm (TA) and Tabu search (TS) for $P|r_j|\sum_j w_j C_j$ Problem. Table (2) show comparison of three local search method when these methods start with good initial solution which is get from two heuristic methods (SWPT) (shorted weight processing times), and (SRD) (shorted release dates). Table (3) like as (table (2)) but the initial solution get from best neighborhood types. In tables (1), (2) we compare the efficiency Memetic algorithm approach (MA), Threshold acceptance algorithm (TA) and Tabu search (TS) have been approached in terms of comparable rate of value (V) and time (T). The Threshold acceptance algorithm (TA) is best value for all job on machine(2), Memetic algorithm approach (MA) is best value for all job on machine(3), and Tabu search (TS) is best value for all job on machine (5). The Threshold acceptance algorithm (TA) is best times for all job on machines (2,3,5).

Table (1) the performance of neighborhood types

M		10	20	30	40	50	75	100	150	200	500	1000	2000	5000
2	INISIAL	419.8667	1494.626	2947.133	5494.133	8142.225	17193.58	30717.26	64728.2	116315.4	726750.8	2909364	11678176	72582050
	Move neigh.	385.6667	1160.68	1604.6	4353.693	6191.35	14059.2	27138.17	62962.06	112701.7	704689.2	2903231	11677237	71057514
	Insert $[i,j]$	384.4667	1062.64	1745.333	4412.1	6503.583	13770.79	27069.47	61966.09	110960.9	700406.2	2896883	11662587	70869639
	SWAP	359.3667	1036.8	1645.033	4374	6155.517	13098.71	26520.57	60411.33	109927.5	683173.6	2891522	11642290	70793631
	Insert $[j,p(l)]$	397.3667	1214.78	2059.633	4872.6	6697.283	14110.25	27324.57	61478.3	110148.5	670836.6	2882477	11583407	70292869
	SWAP 2 JOBS	376.4667	1299.78	2071.633	5040.443	6858.217	15352.01	28276.27	63171.44	113534.9	705998.9	2901707	11674169	71048676
	MIN	359.3667	1036.8	1604.6	4353.693	6155.517	13098.71	26520.57	60411.33	109927.5	670836.6	2882477	11583407	70292869
3	INISIAL	196	647.55	1316.457	2462.246	3519.863	7340.831	13112.1	27298.47	51366.27	314133.1	1256137	5051549	31101460
	Move neigh.	178.3667	592.55	1212.8	2205.92	3423.246	6801.597	12503.39	26539.45	50528.5	313137	1254935	5050047	31099959
	Insert $[i,j]$	185.1167	606.3	1244.75	2310.477	3432.649	7012.554	12710.8	26601.57	50338.45	311706	1249541	5038410	31073853
	SWAP	168.1667	562.5667	1188.737	2229.337	3323.843	6880.185	12348.97	26132.9	49221.11	308766.6	1243895	5022616	31034680
	Insert $[j,p(l)]$	192.9	639.6	1316.457	2454.422	3493.046	7338.364	13092.84	27280.67	51340.84	313781.3	1254078	5044876	31093627
	SWAP 2 JOBS	196	647.55	1316.4	2457.446	3519.863	7340.831	13102.46	27288.62	51353.56	313867.8	1255981	5049061	31096677
	MIN	168.1667	562.5667	1188.737	2205.92	3323.843	6801.597	12348.97	26132.9	49221.11	308766.6	1243895	5022616	31034680
5	INISIAL	123.6175	305.8947	608.329	1041.691	1485.719	2977.988	5235.844	10878.25	19811.85	118829.3	471493.9	1880288	11639250
	Move neigh.	104.1	272.5	540.3833	935.06	1292.867	2702.077	4976.888	10556.23	19720.47	117777.1	469462	1877860	11633737
	Insert $[i,j]$	105.6	274.8	551.1333	976.76	1388.426	2882.625	5137.519	10762.85	19746.56	117887.3	468702.3	1874919	11626720
	SWAP	92.9	247.1143	500.2119	897.985	1289.613	2698.636	4959.031	10327.21	19520.71	115899.2	466032	1868141	11617084
	Insert $[j,p(l)]$	122.33	300.529	599.0729	1041.691	1478	2961.346	5095.643	10871.57	19803.59	116379.3	463187.4	1863179	11590791
	SWAP 2 JOBS	108.9	296.3857	587.9833	1024.093	1469.627	2942.378	5163.131	10802.2	19732.98	117957.8	469765.9	1877247	11629460
	MIN	92.9	247.1143	500.2119	897.985	1289.613	2698.636	4959.031	10327.21	19520.71	115899.2	463187.4	1863179	11590791

Table(2)The performance of three local search method when these methods start with good initial solution which is get from two heuristic methods (SWPT) (shorted weight processing times), and (SRD) (shorted release dates)

M	n	UB	MA		TH		TS	
			VALUES	TIMES	VALUES	TIMES	VALUES	TIMES
2	10	419.86667	359.70667	0.124251	358.16667	0.0174512	359.76667	0.0379077
	20	1494.6257	1055.6397	0.1803118	1030.14	0.0163288	1044.4	0.0377286
	30	2947.1333	1720.5405	0.2333202	1644.0333	0.0167008	1637.5333	0.0398638
	40	5494.1333	4343.7621	0.2880169	4311.05	0.0184245	4321.75	0.0382915
	50	8142.225	6513.2673	0.3428978	6161.8667	0.0174311	6184.7167	0.0394023
	75	17193.583	13516.942	0.4981693	13037.888	0.0180377	13008.525	0.0463554
	100	30717.26	27304.521	0.6480451	26063.5	0.0189265	26151.468	0.0501647
	150	64728.204	61671.034	1.0277303	60763.3	0.0202358	60807.62	0.0442482
	200	116315.41	111381.62	1.4333909	109868.31	0.0211679	110128.16	0.0497723
	500	726750.81	689645.23	4.3859568	680020.5	0.0314309	684657.59	0.0875024
	1000	2909364	2894088.7	11.494485	2891195.5	0.0469107	2891871	0.076835
	2000	11678176	11648584	33.155939	11646832	0.0790935	11640233	0.1609947
	5000	72582050	71207223	147.05215	70804590	0.2005945	70798788	0.8667717
3	10	196	176.83167	0.1397935	187.2	0.0333768	184.85873	0.0341948
	20	647.55	606.34143	0.1910327	616.55	0.0346968	609.39365	0.0359947
	30	1316.4571	1254.8652	0.2456646	1286.444	0.0347889	1285.4971	0.0382817
	40	2462.2462	2314.0217	0.3022341	2313.9725	0.0352639	2357.2446	0.0377739
	50	3519.8625	3447.7517	0.3564184	3475.6701	0.0357993	3516.0375	0.0371936
	75	7340.8306	7064.689	0.5086188	7340.8306	0.0378422	7089.0237	0.0413126
	100	13112.097	12736.945	0.6697063	12979.124	0.0392601	12770.697	0.0433846
	150	27298.473	26948.994	0.977642	26703.539	0.0436202	26823.142	0.0477326
	200	51366.268	50517.881	1.2977759	50409.488	0.0455283	50779.679	0.0514781
	500	314133.13	311925.97	3.8734856	311659.08	0.0634641	312130.2	0.0768899
	1000	1256136.8	1249657.3	10.111567	1251834	0.0906378	1250470.8	0.1227459
	2000	5051548.9	5039866.9	29.546275	5039773.2	0.1578318	5039794.6	0.246138
	5000	31101460	31073414	145.44128	31082299	0.352088	31077981	0.5947781
5	10	123.6175	102.78159	0.1494628	102.9	0.032246	108.40698	0.0475413
	20	305.89468	286.66278	0.2015698	281.84444	0.0317631	285.10079	0.048508
	30	608.32905	571.86667	0.2542034	576.76667	0.0320604	569.0881	0.0493561
	40	1041.6906	990.40266	0.313983	983.63333	0.0319043	996.3375	0.05005
	50	1485.7193	1441.5137	0.3688772	1454.4429	0.0327535	1411.1454	0.0509319
	75	2977.9875	2963.1303	0.5122929	2973.8209	0.0336831	2879.1161	0.0527893
	100	5235.844	5123.8516	0.6518393	5188.2254	0.035814	5098.5749	0.0526805
	150	10878.248	10826.87	0.9538139	10733.468	0.0394088	10751.417	0.0563398
	200	19811.852	19778.033	1.2683401	19781.507	0.0414777	19757.908	0.0595755
	500	118829.34	118356.14	3.8169265	118197.98	0.056248	117920.44	0.0838675
	1000	471493.95	471080.75	10.025859	469151.54	0.0811237	469523.1	0.124201
	2000	1880288.3	1880288.3	29.067407	1876443.9	0.1298392	1876281.8	0.2275431
	5000	11639250	11629878	140.4961	11625462	0.3032346	11630446	0.5405126

Table (3) The performance of local search methods when the initial solution get from best neighborhood types

M	n	Nei	Memetic		TH		TS	
			VALUES	TIMES	VALUES	TIMES	VALUES	TIMES
2	10	359.36667	356.24667	0.1221558	358.06667	0.0160009	359.36667	0.0171704
	20	1018	1004.1978	0.1767784	1005.84	0.0165376	1004	0.0177178
	30	1595.2	1591.8095	0.2257501	1583.9333	0.016837	1563.5	0.0186026
	40	4273.65	4226.2536	0.2838984	4207.2	0.0169905	4212.7	0.0188071
	50	6126.9833	6015.635	0.342817	5976.9667	0.0173553	5984.9833	0.0199491
	75	13098.713	12789.075	0.4792318	12393.75	0.018113	12459.2	0.0219774
	100	26445.368	25891.625	0.6213327	25426.6	0.0187814	25602.468	0.0229412
	150	60411.329	59373.769	0.9251299	59382.523	0.0203839	59281.941	0.0241152
	200	109581.54	108181.48	1.2687709	107429.96	0.0215188	107310.69	0.0265009
	500	670836.57	658977.45	3.8664861	655709.5	0.0328406	654408.77	0.0503392
	1000	2882477.1	2875242.6	10.136502	2875291	0.0518994	2873749.6	0.0595975
	2000	11583407	11543154	28.973093	11546197	0.0808915	11554617	0.1422821
	5000	70292869	70170894	151.55515	70044376	0.1815122	70027064	0.7370488
3	10	166.56667	166.56667	0.1352777	163.96667	0.0223382	163.56667	0.0211848
	20	562.06667	562.06667	0.1913307	559.36667	0.0198184	558.46667	0.0209428
	30	1178.8667	1165.2567	0.282192	1148.4667	0.0203388	1158.0857	0.0213979
	40	2197.22	2184.8394	0.35765	2165.4	0.0205579	2165.4367	0.0213249
	50	3290.93	3235.4317	0.3937103	3154.8733	0.0203634	3199.6417	0.0223792
	75	6757.7656	6689.5841	0.5541724	6517.5776	0.022498	6558.2208	0.0244501
	100	12328.3	12181.85	0.7257782	12010.571	0.0216802	11928.935	0.0265547
	150	26103.07	25730.268	1.0740765	25314.159	0.0231294	25304.198	0.0301014
	200	49221.108	48496.899	1.4598517	47996.184	0.0258494	47709.067	0.0329539
	500	308766.57	306317.81	4.4777277	303740.79	0.0361337	302867.61	0.0567856
	1000	1243895.3	1238788.7	11.569771	1230956.1	0.0505896	1231430.6	0.1160968
	2000	5022615.8	5015381.1	33.171259	4995065.9	0.0836233	4999809.4	0.2362307
	5000	31034680	31010084	157.55845	30971728	0.190418	30973941	0.612971
5	10	90.7	90.7	0.1434232	90.7	0.0262714	90.7	0.0356703
	20	245.3	245.3	0.199091	242.2	0.0255362	241	0.0274463
	30	498.9119	498.9119	0.2558105	488.61667	0.0256621	482.3119	0.0282433
	40	891.525	891.525	0.3088707	860.3	0.0260922	854.22	0.029621
	50	1278.1333	1277.6133	0.3655092	1253.675	0.0269021	1231.5133	0.0301909
	75	2674.2333	2674.2333	0.5080963	2612.6847	0.0273448	2578.0939	0.0316005
	100	4922.781	4918.7095	0.6590784	4779.8583	0.0278559	4826.7643	0.0334803
	150	10326.208	10324.18	0.9664086	10093.588	0.0292771	10047.595	0.0376512
	200	19520.706	19520.706	1.2931839	19371.887	0.0314287	19408.724	0.0369334
	500	115652.16	115637.65	3.8598413	114356.82	0.0400443	114367.84	0.0639215
	1000	463187.38	463187.38	10.101418	460594.55	0.0544169	460923.23	0.1198947
	2000	1862929.4	1862929.4	30.522191	1857918.9	0.0846591	1857405.2	0.2361335
	5000	11590791	11590791	157.09561	11577239	0.186721	11576716	0.6186447

n: Number of jobs
M: Number of machines
MA: Memetic algorithm approach
TA: Threshold acceptance algorithm
TS: Tabu search
Nei: The neighborhood types

7. Future work

Some suggestions for future research are described as follows:

- First, the extensions propose of the exact for $P|r_j|\sum_j\left(w_jC_j/W+h_jT_j/H\right)$ problem by driving a good lower bound or using the dominance rule in branch and bound algorithm.
- Second, using the local search heuristic should be explored finding an improvement potential of various polynomially bounded scheduling heuristic.

Reference

[1] Ahn, B.H, and Hyun, J.H., "Single facility multi class job scheduling", computers and operation research vol.17(1990),265-272.

[2] Bruno, L. Coofman J. and Sethi, R., "Scheduling independent tasks reduce mean finishing time" coounications on Acm vol.17 (1974)382-387.

[3] Cheng, T. and Sin, C., "A state of the review of parallel machine scheduling research" European journal of OR. Vol.47, 271-292.

[4] Franca, P. M., Mendes, A. and Moscato, P., "A memetic algorithm for the total tardiness single machine scheduling problem", European journal of OR. (1999)

[5] Gendreau M., "An introduction to tabu search", in glover F. and Kochenberger G.A. editors handbook of meta heuristic 37-54, Boston Kluwer academic publishers (2003)

[6] Glover F., and Laguna, M., "Tabu search", Boston: Kluwer academic publishers (1997).

[7] Leung, J.Y.-T. Li, Li, H., and Pinedo, M. L., "Scheduling orders for multiple product types to minimize total weighted completion time", Discrete applied.

[8] Ramachandra, G., and Elmaghraby, S. E., "Sequencing precedence related jobs on two machine to minimize the weighted completion time", int. J-production Economics .vol.100, (2006)44-58.

[9] Moscato, P., "On evolution search optimization genetic algorithms and martial arts: towards memetic algorithm", Caltech concurrent computation program c3p roport 826(1989).

[10] Nessah, R., Chu, C., and Yalaoui, F., "An exact method for $P_m|sds,r_j|\sum_j C_j$ problem", Computers and OR.vol.34,(2007)2840-2848.