Ibn Al Haitham Journal for Pure and Applied Science

Journal homepage: http://jih.uobaghdad.edu.iq/index.php/j/index

Genetic Algorithm and Particle Swarm Optimization Techniques for Solving Multi-Objectives on Single Machine Scheduling Problem

Alaa Sabah Hameed **Hanan Ali Chachan**

Department of Mathematics College of Sciences, Al-Mustansireah University, Iraq.

alaasabah127@gmail.com Hanan_altaai@yahoo.com

Article history: Received 29 April 2019, Accepted 11 June 2019, Publish January 2020.

Doi: 10.30526/33.1.2378

Abstract

In this paper, two of the local search algorithms are used (genetic algorithm and particle swarm optimization), in scheduling number of products (n jobs) on a single machine to minimize a multi-objective function which is denoted as $1//\sum_{j=1}^{n}(C_j + T_j + E_j + V_j)$ (total completion time, total tardiness, total earliness and the total late work). A branch and bound (BAB) method is used for comparing the results for (n) jobs starting from (5-18). The results show that the two algorithms have found the optimal and near optimal solutions in an appropriate time.

Key word: The Branch and Bound method (BAB), The Local search algorithms, The Genetic Algorithm (GA), The Particle swarm optimization (PSO), The Multi-Objective problems.

1.Introduction

In the matter of dealing with scheduling problems, there are given a finite or countable infinite set of solutions in which we have to find one solution of them that optimizes (minimizes or maximizes) a given specific cost function. The computational complexity theory shapes the difference between easiness and hardiness of the problems, where a problem is easy if there is a polynomial-time algorithm solves it and obtains an optimal solution. A problem is said to be hard (or NP-hard) if there exist no such polynomial-time algorithm to solve it. So, when we deal with an NP-hard problem, there are two ways for treating it. The first one is to target an optimal solution, while the second is by using a heuristic algorithm. The solutions which are found by the heuristic algorithm do not need to be optimal, but they are found with an acceptable time (i.e. the heuristic algorithms trade off the optimality versus the computing time). The heuristic algorithms can be classified into *constructive algorithms* and *local search algorithms*. The constructive algorithm generates a solution by a number of steps, where in every step the partial solution obtained is extended till we get the complete solution in the last step. On the other hand, the local search algorithms are search in the entire solution space for the best solutions, in other words it begins with an initial solution and then recursively generate a new solution which is near to the current one. While the solutions of many scheduling problems can be represented by sequences, permutations and graphs, the local search algorithms can use these representations by defining

neighborhood functions with conditions of reordering items locally to get (obtain) a neighboring solution [1]. Since the introduction of the local search techniques in the combinatorial optimization problems, the specialists have used these techniques for solving the NP-hard problems. Using these techniques in scheduling makes an allowance for us to test problems with a large number of jobs. Minimizing the total late work on a single machine was treated by Chin-Chia Wu [2]. where he proposed three genetic algorithms and combined them to get the fourth one, his computational results showed that the three (GA) algorithms were getting the stability state when n becomes larger. The problem $1//\sum C_i + V_{max} + E_{max}$ was solved by Tariq and Doha [3]. by using simulated annealing (SA) and descent method (DM) for (75,…,30000) jobs, and they show that the (SA) gives a reasonable results for small n, and the times for both (DM) and (SA) algorithms are equal. Tariq and Faez [4]. propose the (PSO) and the (GA) as heuristic methods to find approximation solutions for $1//Lex\ (\sum C_j\ , \sum T_j)$ and they found that these local search algorithms solve the problem for $n \leq 2000$ jobs with reasonable time. The (PSO) algorithm were applied by Hanan [5]. For solving the problem $1//\ (\sum W_j C_j + L_{max}^{w} + E_{max}^{h})$ where she proposed a new style of development steps to achieve good convergence in application, and made a comparison between (PSO) and (GA) showed that the results of (PSO) are better than (GA) for $n = 2000$ jobs. Tariq, and Hafed reach $n \leq 10000$ jobs by using three local search techniques; descent method (DM), simulated annealing (SA) and tabu search (TS) in solving the problem $1//\ (\sum C_j + \sum T_j + T_{max} + E_{max})$, where they showed that the performance of the algorithms is evaluated on a large set of test problems and the results which are compared showed that (SA) and (TS) algorithms are better than (DM) with preference to (SA) algorithm, and showed that the three algorithms find optimal or near optimal solutions in a reasonable times [6].

2.Problem Representation

A multi-objective problem is considered, and the formal description of this problem is set as follows:

Scheduling n jobs on a single machine which is always available can execute them, where each one of these jobs can be executed on that machine at its special time (i.e. only one job can be executed at a time), and the machine can do only one job at a time.

For $j = 1,2,3, \ldots n$ we will denote p_j and d_j as the processing time and the due date of the j^{th} job respectively. The schedule (π) will define a completion time $C_j(\pi) = \sum_{i=1}^{j-1} p_i(\pi) + p_j(\pi)$ for every job j. The tardiness $T_j(\pi) = \max\{C_j(\pi) - d_j(\pi), 0\}$ and the earliness $E_j(\pi) = \max\{d_j(\pi) - C_j(\pi), 0\}$ will show up for every job j. The late work $V_j(\pi)$ of job j is the amount of the processing time $p_j(\pi)$ that is performed after the due date $d_j(\pi)$, where if $(V_j(\pi) = 0)$ then j is early, if $(0 < V_j(\pi) < p_j(\pi))$ then j is partial early and if $(p_j(\pi) \leq V_j(\pi))$; then j is late [7]. Every job j will be ready to be processed at time zero, where no preemption is allowed and our objective is to find a feasible solution that gives the minimum value of the multi-objective function $F = \sum_{j=1}^{n}(C_j(\pi) + T_j(\pi) + E_j(\pi) + V_j(\pi))$.

Using the standard scheduling problem classification notation, our problem is denoted by $1|\ |\sum_{j=1}^{n}(C_j(\pi) + T_j(\pi) + E_j(\pi) + V_j(\pi))$ and formulated as;

$$\left.\begin{array}{ll} Min\ F(\pi) = Min \sum_{j=1}^{n} (C_j(\pi) + T_j(\pi) + E_j(\pi) + V_j(\pi)) & \\ subject\ to: & \\ C_j(\pi) = p_1(\pi)\ ; & j = 1 \\ C_j(\pi) = C_{j-1}(\pi) + p_j\ (\pi); & j = 2, \dots, n \\ T_j(\pi) = \max\{C_j(\pi) - d_j(\pi), 0\}, & j = 1, \dots, n \\ E_j(\pi) = \max\{d_j(\pi) - C_j(\pi)\ , 0\}, & j = 1, \dots, n \\ V_j(\pi) = \begin{cases} 0 & if \quad C_j(\pi) \le d_j(\pi) \\ C_j(\pi) - d_j(\pi) & if\ d_j\ (\pi) < C_j(\pi) < d_j(\pi) + p_j(\pi) \\ p_j(\pi) & if \quad d_j(\pi) + p_j(\pi) \le C_j(\pi) \end{cases}, & j = 1, \dots, n \end{array}\right\} \quad (A)$$

3. Heuristics for the Problem

In this section, we will mention the heuristics that are used for the problem (A) where there are two simple known heuristics used as an initialization for looking at feasibility of the solutions in the entire search space.

3.1. First Heuristic (H1): This heuristic is obtained by applying the shortest processing time (SPT) rule (i.e. sorting the jobs in order of $(p_1 \le p_2 \le \cdots \le p_n)$).

3.2. Second Heuristic (H2): This heuristic is obtained by applying the earliest due date (EDD) rule (i.e. sorting the jobs in order of $(d_1 \le d_2 \le \cdots \le d_n)$).

4. (BAB) Method

The (BAB) method depends basically on the complete enumeration in the search area. It consists of two procedures; branching and bounding. The branching procedure is the dividing of a large problem into two or more sub-problems, while the bounding calculates a lower bound on the optimal solution's value for every sub-problem [8].

4.1 Upper Bounds: As initialization of searching in the search tree by using the (BAB) method, the two heuristics in (3.1) and (3.2) are used to play as the upper bounds of our problem in this paper.

4.2. Lower Bound: For deriving a lower bound, the problem (A) can be decomposed into two sub-problems$(A_1)\ and\ (A_2)$, where:

$$\left.\begin{array}{ll} Min\ F_1 = Min \sum_{j=1}^{n} \big(C_j(\pi) + T_j(\pi) + E_j(\pi)\big) & \\ \boldsymbol{subject\ to}: & \\ C_j(\pi) = p_j(\pi), & for\ j = 1 \\ C_j(\pi) = C_{j-1}(\pi) + p_j(\pi), & for\ j = 2, \dots, n \\ T_j(\pi) = \max\{0, C_j(\pi) - d_j(\pi)\}, & for\ j = 1, \dots, n \\ E_j(\pi) = \max\{0, d_j(\pi) - C_j(\pi)\}, & for\ j = 1, \dots, n \end{array}\right\} \quad (A_1)$$

For this sub-problem, the lower bound, which was applied by Hussam Abid Ali [9]. is used to obtain the first lower bound (LB_1), where:

$$\sum_{j=1}^{n}\big(C_j + T_j + E_j\big) = Max\{\sum_{j=1}^{n} d_j, \sum_{j=1}^{n} Max\ \{2C_j - d_j\ , C_j\}\} \quad (1)$$

$$\left.\begin{array}{ll} Min\ F_2 = Min \sum_{j=1}^{n} V_j(\pi) & \\ subject\ to: & \\ C_j(\pi) = p_j(\pi), & for\ j = 1 \\ C_j(\pi) \geq p_j(\pi), & for\ j = 2, \dots, n \\ T_j(\pi) = \max\{0, C_j(\pi) - d_j(\pi)\}, & for\ j = 1, \dots, n \\ V_j(\pi) = \min\{p_j(\pi), T_j(\pi)\}, & for\ j = 1, \dots, n \end{array}\right\} \quad (A_2)$$

Here, the lower bound in the theorem (4.1) below is used for (A_2) to get the second lower bound (LB_2).

Theorem 1[7].

If $p_{max} = \max\{p_j\}$ and $T_{max} = \max\{T_j\}$ where$j = 1, 2, \dots, n$, then we have that;
$T_{max} \leq \sum_{j=1}^{n} V_j \leq min\ \{nT_{max}\ ,\ T_{max} + p_{max} - 1\}$.

Now, the following lemma allows us to use $LB = LB_1 + LB_2$ as the lower bound for the problem (A).

Lemma 1 [10].

If (LB_1) and (LB_2) are the lower bounds of the problems $(A_1)\ and\ (A_2)$ respectively, then $= LB_1 + LB_2$, is the lower bound of the main problem(A).

4.2.1. (LB) procedure

For N, S and U where N represents the set of all jobs, S is equal to the set of the scheduled jobs and U is the set of the un-scheduled jobs, then the procedure is:

1.Starting with empty set of the scheduled jobs (i.e. $S = \emptyset$), and begin to sort the jobs (one by one) until we have |S|$= n - 1$, and the n^{th} job will be add to the set S then we solve the last sequence by the complete enumerate method (CEM). At every step we calculate the cost $\sum_{j \in S}(C_j + T_j + E_j + V_j)$.

2. For the set U, the jobs have been sorted in two rules for calculating the costs for the two sub-problems $(A_1)\ and\ (A_2)$ by doing the following steps;

Step (1): Sorting the jobs in the set U by (SPT) rule, and then calculate $\sum_{i \in U}(C_i + T_i + E_i)$ by using equation (1).

Step (2): Re-sorting the jobs in the set U by (EDD) rule, and calculate $(\sum_{i \in U} V_i)$ by using theorem (4.1).

Step (3): Calculate the total cost $\sum_{j \in N}(C_j + T_j + E_j + V_j)$ as follows:

$$\text{Total cost} = \sum_{j \in S}(C_j + T_j + E_j + V_j) + \sum_{i \in U}(C_i + T_i + E_i) + \sum_{i \in U} V_i \quad (2)$$

5. Local Search Methods

In a matter of using the local search methods, there is no guarantee of obtaining optimality, but using them may give us solutions that are near the optimal. Therefore, the local search methods considered as the second choice of solving the NP-hard problems. In this paper, two of these methods are applied the genetic algorithm (GA) and the particle swarm optimization (PSO).

5.1. Genetic Algorithm (GA):

The genetic algorithm (GA) is an evolutionary search technique used for the scheduling problems to obtain a near optimal solution for complex problems [11]. It begins with a randomly

generated population of chromosomes (feasible solutions) and replaces (iteratively) this population with a new one. The (GA) requires for the problem a good representation, and a fitness function that measures the chromosome's quality. Regeneration technique depends on selection of ancestors (parents) and reunites them by using the crossover to get successor (children), then applying the mutation to change them (locally) for obtaining better results [12].

5.1.1. (GA) Operators

The (GA) has a number of operators:

1. Representation: In this paper, a chromosome is represented by a sequence of jobs where every gene refers to a job [13].

2. Initialization: The initial population can be obtained by either *introducing heuristics* or random arranging [13]. In our paper, we choose the first way (i.e. introducing heuristics), and we take 50 Chromosomes as the size of our initial population as follows: create an initial population of (50) chromosomes, choose five chromosomes where three of them are randomly selected while the remain two chromosomes by applying the earliest due date (EDD) and the shortest processing time (SPT) (i.e. seeding good parents).

3. Selection: We use the roulette wheel selection method, where we choose the chromosome with the lowest fitness value since it has a higher probability of participating in one or more children to the new generation.

4. Fitness function: The fitness function specifies a value reflects the quality (goodness or badness) of the chromosome [14]. Here, in this paper, the considered function is;

$$1//\sum_{j=1}^{n}(C_j + T_j + E_j + V_j).$$

5. Crossover: The 1-point legitimate crossover (LEGX) [15]. Is used. The cut probability is 0.2 when $n \leq 5000$ and when $n > 5000$ we cut at every 1000 jobs. In parent 1 the cut will be at the end, while in the parent 2, the cut will be at the beginning, as the following example;

Parent (1): 6 4 5 2 0 | 9 8 7 1 3 ⟹ Child (1): 6 4 5 2 0 | 9 8 7 3 1

Parent (2): 0 2 3 9 4 | 5 1 7 6 8 ⟹ Child (2): 2 0 3 9 4 | 5 1 7 6 8

Where we generate the new population by mating each chromosome from step 2 with the whole five chromosomes, and every parent chromosome will produce 10 children chromosomes, so the resulting new population will consists of 50 new chromosomes.

6. Mutation: In getting better results, the mutation operator inside the sequence as an intent to get an improvement [16]. For our problem, the random swap between jobs is applied.

7. Termination: The stopping criterion is 600 seconds.

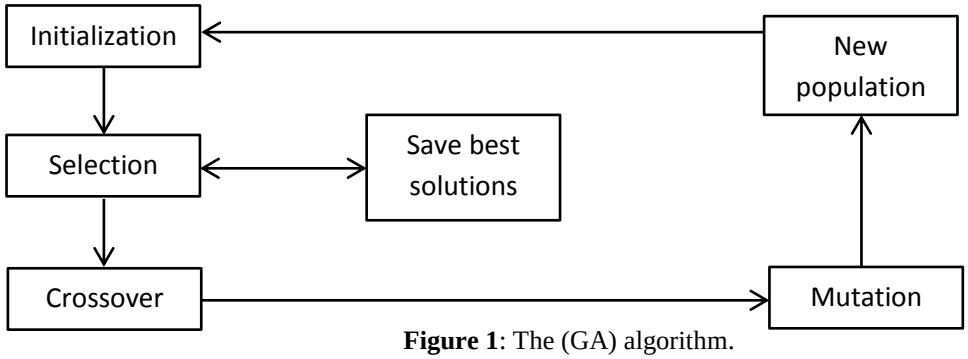

Figure 1: The (GA) algorithm.

5.2. Particle Swarm Optimization (PSO):

The (PSO) is a very simple algorithm and effective optimizer for functions from wide range developed by Eberhart [17]. It is based on social simulation models. This algorithm assigns a collection (population) of search points (solutions) moving randomly in the search space, and the best position found by every individual which is also said to be is saved in *memory*. Then, this experience of the individual is connected with the part or whole of the swarm to change the movement direction to the best locations were found till now. The swarming behavior is produced by employing main rules which are; *the velocity matching and acceleration by the distance* applying by every individual in the swarm in their searching of food [18]. Where:

$$V_{ij}(t+1) = \omega * v_{ij}(t) + \alpha\, \mathcal{F}_1\left(p_{ij}(t) - x_{ij}(t)\right) + \beta\, \mathcal{F}_2\left(p_{gj} - x_{ij}(t)\right) \quad (3)$$

$$X_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \quad j = 1,2,\dots,L\,. \quad (4)$$

Where $\boldsymbol{\omega}$ is the inertia parameter, $\mathcal{F}_1$ & $\mathcal{F}_2$ are two arbitrary functions of the (0, 1) range, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are two constants, the swarm (or population) is a set of (Y) particles positioned in L-dimensional space. At every iteration t (where t =1,…,T) , the i^{th} particle (i=1,…,Y) has a position $X_i(t) = (x_{ij}(t))$ where j=1,…, L and a velocity $V_i(t) = v_{ij}(t)$. The velocity $V_i(t+1)$ is the rate of moving the i^{th} particle from $X_i(t)$ to $X_i(t+1)$. Every position $X_i(t)$ may be (directly or indirectly) represent a solution for a problem. The objective function is denoted as $f(X_i(t))$, which is also called the fitness function. In the minimization problem (as in this paper) X_a is better than X_b (where a, b $\in Y$) when $f(X_a) < f(X_a)$. The i^{th} particle pest position is the position when the function $f(X_i(t))$ takes the minimum value till now and denoted as $P_i = (p_{ij})$. The global best position ($\boldsymbol{g_{best}}$) is denoted as $P_g = (p_{gj})$ which is the best position found by the entire swarm. [19]. Then the algorithm is:

5.2.1. (PSO) algorithm [17].

Step 1: Initialize the population with 7 particles in which have random positions and velocities.

Step 2: Evaluate the objective (the fitness) function for each particle in the swarm to get the best solution s^*.

Step 3: From s^* we generate the new population of 7 other solutions, as follows:

- s_1: Swapping two jobs in the sequence of s^*.
- s_2: Swapping two jobs (deferent than s_1) in s^*.
- s_3: Swapping two jobs (deferent than s_1 and s_2) in s^*.
- s_4: Ordering the first half of s^* in EDD rule.
- s_5: Ordering the second half of s^* in EDD rule.
- s_6: Ordering the first half of s^* in SPT rule.
- s_7: Ordering the second half of s^* in SPT rule.

Step 3: Compare the particle's fitness evaluation with its p_{best}, if the $current\ value < p_{best}$ then set: ($p_{best} = current\ value$) and ($P_i = current\ position$)**.**

Step 4: Compare evaluation with g_{best}, if current value $< g_{best}$ then set P_g= particle's index.

Step 5: Update the velocity by (3) and the position by (4).

Step 6: Go back to step 2.

6. Experimental Results

6.1. The Problems Instances:

The performance of the (BAB) procedure is compared on 5 problem instances, the sizes of these examples are n = [5, 18]. The problem instances were generated randomly, and for each job j where $j \in \{1, \dots, n\}$, the processing time p_j was uniformly generated in [1, 10]. While the due date d_j was uniformly generated in the interval $[(1 - T - RDD/2)TP, (1 - T + RDD/2)TP]$ as it has been showed in the literature [20]. Where $T = \sum_{j=1}^{n} p_j$ and the two parameters (TP and RDD) are said to be the tardiness factor and related range of due dates respectively, and have the following values:

$RDD = 0.2\,, 0.4\,, 0.6\,, 0.8\,, 1$ and $TP = 0.2\,, 0.4$

6.2. Computational Results

In this subsection, the computational results is given in tables, each table of them gives the results. In **Table 1.** we put the comparison among BAB, GA and PSO for n = [5, 18]. The **Table 2.** is contained the values and times by averages for n = [50, 20000]. For each n there is 5 problems examples are tested. The symbols which used in the tables are:

n: The number of jobs,

BAB: The branch and bound method,

GA: The genetic algorithm,

PSO: The particle swarm optimization,

$V_{avg.}$**:** The average of the value**,**

$T_{avg.}$**:** The average of the execution time of the problem (by second).

Best: The best (value & time) average**.**

No. : The number of (*) and the number of (#).

Note: The symbol (*) refers to the minimum value's average, and the symbol (#) refers to the minimum time's average of each (n).

6.3. The Tables of Results

In **Table 1.** the results of applying (BAB, GA and PSO) are showed for n= [5, 18]. Jobs. For each n there are 5 different examples are tested. These results showed that the value averages for n={5,6,7,8,9,10,11,12,15} of using (BAB and GA) are equal, while the averages of using (PSO) are bigger with small differences. The execution time's average results showed the priority of (GA) among them (i. e. the (GA) is faster than the others).

Table 1: comparison among BAB, GA and PSO.

N	*BAB*		*GA*		*PSO*		*Best*	
	$V_{avg.}$	$T_{avg.}$	$V_{avg.}$	$T_{avg.}$	$V_{avg.}$	$T_{avg.}$	$V_{avg.}$	$T_{avg.}$
5	124.2*	0.0777	124.2*	0.0367#	124.2*	0.0949	124.2	0.0367
6	145*	0.0877	145*	0.0422#	145*	0.0975	145	0.0422
7	184.4*	0.1559	184.4*	0.0430#	184.4*	0.0900	184.4	0.0430
8	343*	0.1998	343*	0.0421#	343*	0.1145	343	0.0421
9	327.6*	0.2573	327.6*	0.0608#	328.8	0.1205	327.6	0.0608
10	432*	0.9539	432*	0.0464#	433.8	0.0979	432	0.0464
11	469.6*	1.8399	469.6*	0.0415#	469.8	0.0903	469.6	0.0415

12	658.6*	2.6089	658.6*	0.0530#	659.6	0.0894	658.6	0.0530
13	905*	11.4598	907.8	0.0370#	910.2	0.0896	905	0.0370
14	773.2*	26.6837	773.6	0.0541#	774.4	0.1144	773.2	0.0541
15	905*	109.5314	905*	0.0456#	909.6	0.1090	905	0.0456
16	1053.4*	884.5501	1053.8	0.0383#	1059.4	0.1078	1053.4	0.0383
17	1147.8*	796.4961	1153.4	0.0470#	1153.4	0.1008	1147.8	0.0470
18	1298.2*	1.2675	1300.6	0.0528#	1306.4	0.1100	1298.2	0.0528
No.	14(*)	0(#)	9(*)	14(#)	4(*)	0(#)		

In **Table 2.** For each (n) there are (5) problems examples for testing. The **Table 2.** Begins with n = 50(50)100, 100(100)1000, 1000(1000)5000, 5000(5000)20000. The results showed that value's averages of (GA) are better than the (PSO's) averages except when (n=600) jobs. The execution time's average results showed that the (PSO) is faster than (GA) in all problems which are tested.

Table 2. the averages of the (values & times) of (GA) and (PSO).

N	***GA***		***P.SO***		***Best***	
	$V_{avg.}$	$T_{avg.}$	$V_{avg.}$	$T_{avg.}$	$V_{avg.}$	$T_{avg.}$
50	8.7386*	0.2869	8.8758	0.0931#	8.7386	0.0931
100	3.2820*	0.5605	3.3680	0.1092#	3.2820	0.1092
200	1.2328*	1.5239	1.3286	0.1832#	1.2328	0.1832
300	3.0913*	3.0638	3.1601	0.2609#	3.0913	0.2609
400	560652*	5.0216	5.7567	0.3218#	560652	0.3218
500	7.8883*	7.6416	8.4154	0.4022#	7.8883	0.4022
600	2.2137	10.7515	1.2308*	0.4589#	1.2308	0.4589
700	1.5772*	14.3291	1.6959	0.5702#	1.5772	0.5702
800	1.9744*	18.4993	2.1872	0.6434#	1.9744	0.6434
900	2.8095*	23.3777	2.8550	0.6884#	2.8095	0.6884
1000	3.2910*	31.1993	3.4474	0.7588#	3.2910	0.7588
2000	1.2931*	111.7079	1.3696	1.5034#	1.2931	1.5034
3000	3.0102*	264.6768	3.1087	2.4566#	3.0102	2.4566
4000	5.1718*	537.8838	5.5883	4.0436#	5.1718	4.0436
5000	8.5145*	603.6106	8.8628	3.6652#	8.5145	3.6652

10000	3.4796*	617.5982	352058926	7.5153#	3.4796	7.5153
15000	7.8295*	632.2623	7.8615	11.4529#	7.8295	11.4529
20000	1.3522*	628.5472	1.3655	15.4144#	1.3522	15.4144
No.	17(*)	0(#)	1(*)	18(#)		

7. Conclusions

In solving the problem A, two local search methods (GA and PSO) are used, The results showed that the two methods have a good efficiency in finding the optimal or near optimal solutions comparing with the (BAB) method which is used, where (GA) showed that it has reached the optimal solutions many times as well as the (BAB), and the (PSO) showed the advantage in the execution times, where it is faster than (GA) in solving the problems.

References

1. Michiels, W.; Aarts, E.; Korst, J. *Theoretical Aspects of Local Search*. ISBN-13 978-3-540-35853-4. Springer. Berlin, Heidelberg. New York.**2000**.
2. Wu, C.; Yin, Y.; Wu, W.; Chen, H.; Cheng, S. Using a branch-and-bound and a genetic algorithm for a single-machine total late work scheduling problem. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*.**2016**, *20*, *4*, 1329–1339.
3. Abdul-Razaq, T.S.; Abbas, D.A. A Function of Two or Three Cost Criteria to Be Optimized. *Al- Mustansiriyah Journal Science*.**2013**, *24*, *2*, 113-132.
4. Abdul-Razaq, T.S.; Ali, F.H. Algorithms for Scheduling a Single Machine to Minimize Total Completion Time and Total Tardiness**.** *Basrah Journal of Science*.**2016,** *34*, *2*, *113-132*.
5. Chachan, H. A. Solving Machine Scheduling Problem Using Particle Swarm Optimization Method. *The Iraqi magazine for managerial sciences*.**2012**, *8*, *33*, 197-213.
6. Abdul-Razaq, T.S.; Motair, H.M. Solving Composite Multi objective Single Machine Scheduling Problem Using Branch and Bound and Local Search Algorithms. *Al-Mustansiriyah Journal of Science*.**2017**, *28*, *3*, 200-208**.**
7. Potts, C.N.; Van Wassenhove, L.N. Single Machine Scheduling to Minimize Total Late Work. *Operations Research*.**1992**, *40*, *3*, 586-595.
8. Blazewicz, J.; Ecker, K.H.; Pesch, E.; Schmidt, G.; Weglarz, J. Handbook on Scheduling: From Theory to Applications. *Springer. Berlin. Heidelberg. New York*, **2007**.
9. Mohammed, H. A. Genetic and Local Search Algorithms as Robust and Simple Optimization Tools**.** M. Sc. thesis. *University of Al-Mustansiriyah, College of Science, Department of Mathematics*, **2005**.

10. Ramadhan, A.M. Single machine scheduling using branch and bound techniques. M. Sc. thesis. *University of Al-Mustansiriyah, College of Science, Department of Mathematics*, **1998**.
11. Chen, Z.; Tsai, C.; Eberle, W.; Lin, W.; Ke, S. Instance selection by genetic-based biological algorithm. *Soft Computing*.**2014**, *19*, *5*, 1269–1282.
12. Essafi, I.; Mati, Y.; Dauzere-Peres, S. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research.***2008**, *35*, *8*, 2599 – 2616.
13. Etiler, O.; Toklu, B.; Atak, M.; Wilson, J. A genetic algorithm for flow shop scheduling problems. *Journal of the Operational Research Society.***2004**, *55*, 830–835.
14. Iyer, S.K.; Saxena, B. Improved genetic algorithm for the permutation flowshop scheduling problem. *Computers & Operations Research*.**2004**, *31*, *593–606*.
15. Reeves, C. R. A genetic algorithm for flow shop sequencing. *Computers and Operations Research.***1995**, *22*, *1*, 5-13.
16. Nordstorm, A.; Tufekci, S. A Genetic Algorithm for the Talent Scheduling Problem. *Computers and Operations Research*.**1994**, *21*, *8*, 927-940.
17. Eberhart, R.; Kennedy, J. A new optimizer using particle swarm theory. *In Proceedings of the 6th Symposium on Micro Machine and Human Science, Nagoya, Japan*.**1995**, *39-43***.**
18. Konstantinos, E.P.; Michael, N.V. *Particle Swarm Optimization and Intelligence: Advances and Applications*. Information science reference, Hershey, New York,**2010**.
19. Pongchairerks, P. Particle swarm optimization algorithm applied to scheduling problems. *ScienceAsia*.**2009**, *35*, *89–94.*
20. Arroyo, C.; Elias, J.; Ottoni, R.; Oliveira, A. Multi-Objective Variable Neighborhood Search Algorithms for a Single Machine Scheduling Problem with Distinct Due Windows. *Electronic Notes in Theoretical Computer Science*.**2011**, *28*, *15–19.*