Journal of Kerbala University , Vol. 12 No.3 Scientific . 2014

Comparison of Genetic Algorithm and Memetic Algorithm for Bicriteria Permutation Flowshop Scheduling Problem

المقارنة بين الخوارزمية الوراثية وخوارزمية الممتيك لدالة ثنائية الاهداف لمسألة الجدولة التبادلية الانسيابية

Ghassan Adnan Khtan[1a], Viean Abdul Muhsin Al-Salihi[2], Mohamed Saleh Mehdi[1b], Hussam Abid Ali Mohammed[1c]
[1]Department of Mathematics, College of Education for Pure Sceince, University of Kerbala,
[2]School of Applied Sciences, University of Technology, Baghdad, Iraq.
[1a]ghassanmath@yahoo.com, [1b]moh.saled81@yahoo.com,
[1c]hussammath5@gmail.com

Abstract

Flowshop scheduling is a well-known research field for many years. As the problem size gets bigger, an analytical solution becomes impossible. Here, heuristic solutions come to the stage. In the literature, generally solutions regarding a multi-objective are developed; and multi-objective is generally used for three machines. In this paper, the weighted mean completion times and weighted mean tardiness flowshop machine scheduling have been considered, so heuristic methods have used: Genetic Algorithms (*GA*) are a population-based Meta heuristics. They have been successfully applied to many optimization problems. However, such pure genetic algorithms that makes them incapable of searching numerous solutions of the problem domain. A Memetic Algorithm (*MA*) is an extension of the traditional genetic algorithm. That uses a local search technique to reduce the Variable Neighborhood Search (*VNS*). The methods were tested and gave various experimental results which shows that a pure memetic algorithm performs better than the pure genetic algorithms for such type of NP-Hard combinatorial problem. And the hybrid genetic algorithms versions with *VNS*, give good solutions better than hybrid *MA* and both were better than pure algorithms.

المستخلص

جدولة المسألة الانسيابية من مجالات البحوث المعروفة منذ عدّة سَنَوات. وكلما كبر حجم المسألة، يصبح الحل التحليلي لها مستحيل. في مثل هذه الحالات نستخدم الحلولَ التنقيبية. عموماً في البحوث السابقة طورت الحلول المتعلقة بالمسائل المتعددة الأهداف؛ وهذه المسائل استخدمت بشكل عام لثلاث مكائنِ. في هذا البحث، تناولنا جدولة (متوسط الأهمية لوقت الاتمام و متوسط الأهمية للتأخير) لثلاث مكائن انسيابية و استخدمنا طرق تنقيبية لإيجاد الحل: ان الخوارزميات الوراثية (*GA*) مع اساس مجتمع سكاني و التي تكون متعددة التنقيب قد طبقت بنجاح على العديد من مسائل الأمثلية، مع ذلك مثل هذه الخوارزمياتِ الوراثية الصافية تجعلها عاجزة عن ايجاد حلول متعددة لهذه المسألة وان الخوارزمية (*MA*) هي امتداد للخوارزمية الوراثية التقليدية التي تستخدم تقنية البحثِ المحليّ لتقليل قيمة دالة الهدف بواسطة بحث الجوار المتغير (*VNS*). هذه الطرق اختبرت وأعطت نتائج مختلفة والتي اظهرت خوارزمية (*MA*) الصافية تعطي نتائج افضل من خوارزمية (*GA*) الصافية ،لمثل هذا النوع من مسائل الـ NP-Hard والتي تكون حساباتها معقدة. وان الخوارزميات الوراثية الهجينة مع (*VNS*) تعطي حلول جيدة افضل من (*MA*) الهجينة و كلتاهما كانتا افضل من الخوارزميات الصافية.

Keywords: scheduling, flowshop, memetic algorithm, genetic algorithm

1 Introduction

In the context of manufacturing, scheduling is fundamentally related to the problem of finding a successive assignment of limited resources to a number of jobs which is optimal in terms of certain performance measures. On many occasions in manufacturing environments, a set of processes are needed to be serially performed in several stages before a job is completed. Such systems are referred to as the flowshop environments. In a flowshop system, a set of n different

jobs needs to be processed on a sequential set of m machines. That is, each job consists of m operations where each operation must be performed on a different machine for an amount of processing time. Each machine can handle only one job at a time and the operation of a machine on a job usually cannot be preempted.

In flowshop scheduling, the processing routes are the same for all the jobs (Solimanpur et al., 2004). In the permutation flowshop, passing is not allowed. Thus the sequencing of different jobs that visit a set of machines is in the same order. In the general flowshop, passing is allowed. Therefore, the job sequence on each machine may be different (Pinedo, 1995).

The multi-objective flowshop scheduling problem has been addressed by some papers on scheduling. Marett and Wright (1996) compared the performance of simulated annealing and tabu search by using them for solving a large and complex multi-objective flowshop problem. Sayin and Karabati (1999) dealt with the scheduling problem in a two machine flowshop environment by minimizing makespan and sum of completion times simultaneously. Danneberg et al. (1999) addressed the permutation flowshop scheduling problem with setup times where the jobs are partitioned into groups or families. Jobs of the same group can be processed together in a batch but the maximum number of jobs in a batch is limited. The setup time depends on the group of the jobs. They proposed the makespan as well as the weighted sum of the completion times of the jobs as objective function. For solving such a problem, they proposed and compared various constructive and iterative algorithms. Toktas et al. (2004) considered the two machine flowshop scheduling by minimizing makespan and maximum earliness simultaneously. Cheachan et al. (2010) proposed a multi-objective algorithm for flowshop scheduling where a minimizing makespan and maximum tardiness was used. Ravindran et al. (2005) proposed three heuristic algorithms for solving the flowshop scheduling problem by makespan and total flow. Loukil et al. (2005) proposed multi-objective simulated annealing algorithm to tackle the multi-objective production scheduling problems (one machine, parallel machines and permutation flowshops). They considered seven possible objective functions (the mean weighted completion time, the mean weighted tardiness, the mean weighted earliness, the maximum completion time (makespan), the maximum tardiness, the maximum earliness, the number of tardy jobs). They claimed that the proposed multi-objective simulated annealing algorithm is able to solve any subset of seven possible objective functions.

In this paper, we deal with a multi-objective permutation flowshop scheduling problem. The weighted mean completion time and weighted mean tardiness are to be optimized simultaneously. To tackle this problem, an effective multi-objective Genetic Algorithm (*GA*) and Memetic algorithm (*MA*). The remainder of this paper is organized as follows: Section 2 gives the problem definition. In Section 3, the background of *VNS*, *GA* and *MA* and previous works are summarized. The experimental results are provided in Section 4. Finally, Section 5 provides conclusions and the future work.

2 Problem Definition

In this paper, a permutation flowshop problem is considered. The permutation flowshop represents a particular case of the flowshop scheduling problems, having as its goal achieving a schedule for a number of jobs on several machines regarding predetermined objective functions and related constraints.

Consider a hypothetical permutation flowshop scheduling problem in which n jobs are to be processed on m machines where the machines are ceaselessly ready to be used from time zero onwards. Each job consists in m operations and the $i-th$ $(i = 1,\dots,n)$ operation of each job must be processed on machine j ($j = 1, 2, \dots, m$).

At any time, every job can be processed at most on one machine and every machine can process at most one job. One job can start on machine j if it is completed on machine $j-1$ and if machine j is free. In addition, preemption is not permitted; i.e., once an operation is started, it must be completed without interruption.

For the permutation flowshop the operating sequences of the jobs are the same on every machine. That is to say, if one job is at the $i-th$ position on machine 1, then this job will be at the $i-th$ position on all the machines.

Given the known uninterrupted processing time of job i on machine j, p_{ij}, and due date of job i, d_i, and the precedence constraints, the objective is to seek a schedule that minimizes the weighted mean completion time and the weighted mean tardiness of the manufacturing system.

2.1 Weighted Mean Completion Time

The first objective considered is the minimization of the weighted mean completion time. This objective can be calculated by the following expression:

$$\frac{\sum_{i=1}^{n} w_i C_{i,j}}{W} \qquad (1)$$

where $C_{i,j}$ is the completion time for job i on machine j, n is the number of the jobs and w_i is an importance factor related to job i. For instance, it may be equal to a holding cost per unit time. These importance factors are not required to be less than 1. W is the sum total of jobs' weights; that is,

$$W = \sum_{i=1}^{n} w_i \qquad (2)$$

Let $C_{\pi_k,j}$ denote the completion time of the kth job, $(k = 1, 2, \dots, n)$ on machine j in an imaginary permutation $\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$, the completion time of the kth job in this permutation, which is equal to $C_{\pi_k,m}$ can be calculated by the following equations:

$$C_{\pi_1,1} = p_{\pi_1,1}$$
$$C_{\pi_k,1} = C_{\pi_{k-1},1} + p_{\pi_k,1} \qquad k = (2, \dots, n)$$
$$C_{\pi_k,j} = max\{C_{\pi_{k-1},j} + C_{\pi_k,j-1}\} + p_{\pi_k,j} \qquad k = (2, \dots, n) \text{ and } j = (2, 3)$$
$$C_{\pi_k,m} = max\{C_{\pi_{k-1},m} + C_{\pi_k,m-1}\} + p_{\pi_k,m} \qquad k = (2, \dots, n) \text{ and } m = 3$$

For more of the shortcut will write the completion time $C_{\pi_k,j}$ as the following $C_{k,j}$, therefore the equation (1) becomes

$$\frac{\sum_{k=1}^{n} w_k C_{k,m}}{W} \qquad (1')$$

2.2 Weighted Mean Tardiness

Another objective considered is the minimization of the weighted mean tardiness. This objective is due-date based and calculates how due-dates are being met. That is to say, this objective takes into account the due dates that are violated. To calculate the value of this objective, the subsequent expression is used:

$$\frac{\sum_{k=1}^{n} h_k T_{k,m}}{H} \qquad (3)$$

Where n is the number of the jobs, $T_{k,m}$ is the tardiness for job k on machine m and equals to $max\{0, C_{k,m} - d_k\}$ and h_k and H are the same as explained weighted in Section 2.1 equation (2).

It can be easily noticed that the objectives considered are inherently contradicting. To illustrate the point, one should take into account that the optimization of the first objective in a single objective problem is performed regardless of the jobs' due-dates. Hence, the resulting sequences may have large due-dates violations, thus imposing large penalties to the system. On the other hand,

while optimizing the second objective, the goal is to schedule jobs as close as possible to their due-dates. However, the sequence obtained is very likely to cause large penalties to the system due to the fact that this sequence is formed without regard to the job's completion times.

3 Methodology

3.1 Variable Neighborhood Search (VNS)

It is clear to solve scheduling problems one tends to use branch and bound (B&B) or Dynamic programming (DP) to find optimal solutions, however, these approaches has two main disadvantages:

- It is mathematically complex and thus a lot of time to be invested.
- When it concerns *NP*-hard problem, the computational time requirements are enormous for large sized problem.

To avoid these draw backs we can appeal to heuristics methods. In recent year, the improvement in heuristic methods has becomes under the name 'local search heuristic' are implemented on the problem of scheduling n of jobs on three machines to minimize the $\sum_{i=1}^{n}\left(w_i C_{i,m}/W + h_i T_{i,m}/H\right)$ (minimize the weighted mean completion time and weighted mean tardiness). For the representation of solution the natural representation will be used. For each local search method a set of parameter setting is necessary for arriving at high performing algorithm. Conclusions concerning implementation of different setting are discussed.

First we introduce some neighborhoods for a permutation problem, where the step of feasible solutions is given by the set of permutations of n jobs [10].

- **Jump (Ju)** In a permutation $\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$, select an arbitrary job π_i and jump it to a smaller position j, $i > j$, or to a large position k, $k > i$. Thus, we have $|N(\pi)| = (n-1)^2$.
- **Pairwise Interchange (PI)** In a permutation π select two arbitrary jobs π_i and π_j, $i \neq j$ and interchange them, and $|N(\pi)| = n(n-1)/2$.
- **Adjacent Pairwise Interchange (API)** This is a special case of both the jump and the pairwise interchange neighborhood. In a permutation π, two adjacent jobs π_i and π_{i+1}, $(1 \leq i \leq n-1)$ are interchanged to generate a neighbor π, where $|N(\pi)| = (n-1)$.
- **Search Dynamic Programming (Dyna)** In this move we composed of a set of independent interchange moves; each such move exchange the jobs at positions i and j, $i \neq j$. Two interchange moves are independent if they don't overlap, that is if for two moves involving position i, j and k, l we have that $min\{i,j\} \geq max\{k,l\}$ or vice versa.

Now, we propose algorithm AH which is applied at VNS to provide a best solution.

3.1.1 Algorithm AH [10]

Step (1) Select an initial solution $\pi_{Ini} = \{\pi_1, \pi_2, \dots, \pi_n\}$ obtain from the arbitrary sequence and calculate objective value of π_{Ini} say $f(\pi_{Ini}) = f_{Ini}$.

Step (2) In this step will be change the initial sequence π_{Ini} by the others neighborhoods and calculate values function for every one i.e.

a. For the neighbor *Ju*, have π_{Ju} and f_{Ju}.

b. For the neighbor *PI*, have π_{PI} and f_{PI}.

c. For the neighbor *API*, have π_{API} and f_{API}.

d. For the neighbor *Dyna*, have π_{Dyna} and f_{Dyna}.

Step (3) Now choose $f^* = min\left(f_{Ju}, f_{PI}, f_{API}, f_{Dyna}, f_{Ini}\right)$ and

$\pi^* = min\left(\pi_{Ju}, \pi_{PI}, \pi_{API}, \pi_{Dyna}, \pi_{Ini}\right)$, then set $f_{Ini} = f^*$ and $\pi_{Ini} = \pi^*$.

Now, we give details about local search methods (*Meta-heuristic methods*) which are used to solve $F3 \parallel \sum_{i=1}^{n}\left(w_i C_{i,m}/W + h_i T_{i,m}/H\right)$ problems.

3.2 Genetic Algorithm Approach

The genetic algorithm (GA) is an optimization and search technique based on the principles of genetics and natural selection. A GA lets a population made up of many individuals to evolve under specified selection rules to a state that maximizes the "fitness" (i.e., maximizes the benefit function).

The figure below [11] shows a basic model of a genetic algorithm, one of the main techniques in artificial evolution. From an initial population (population of parents). Crossover and mutation are possible to give children. Until the evolution is stopped, children are selected to become parents and so on. This basic model can be modified to match the requirements of the problem to solve.

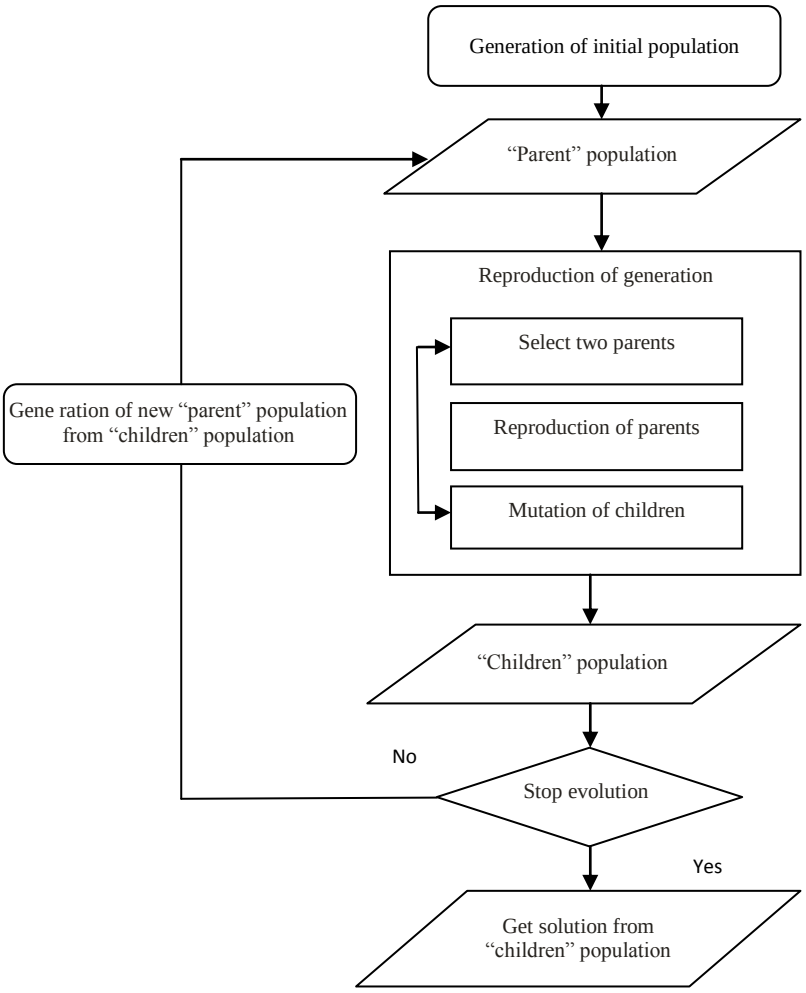


Fig.: Basic genetic algorithm

Proposed genetic algorithm: in this research, the chromosome or the individual that stands for a solution has two main components: the sequence itself and the idle times inserted at the beginning of the schedule. For example [5] [4,1,2,3] indicates that the processing order is: jobs 4, 1, 2 and 3, with first job starting at time 6. The genetic operators used were:

- **Crossover**: Two different crossover operators were implemented. The first is the well-known Order crossover (OX) [10]. After choosing two parents, a fragment of the chromosome from one of them is randomly selected and copied in to the offspring. In the second phase, the offspring's empty positions are sequentially filled according to the chromosome of the other parent.

  The second crossover calls homogeneous mixture crossover (HMX) was proposed by Mohammed et al. [12], given by the mixture the two chromosomes from parents uniformly by making a set from genes M, they introduced the way for the mixture, first; the odd position from

the first parents and the even position from the second parents. Then separate genes without repetition gene, since we read the set M from the left, if the gene *j* does not existing in the first child put it, otherwise we put gene *j* in the second child until final M. This way also gives a new two chromosomes.

- **Mutation**: In our implementation a traditional mutation strategy based as indicated at (3.1.1 Algorithm AH). According to it, we choose minimum value from AH.

Outline of the Basic Genetic Algorithm (GA):

- **Initialization**: In the first step of GA many individual solutions are randomly generated to form an initial population. The population initial generation depends on the nature of the problem, but typically contains several hundreds of possible solutions. Traditionally, the initial population is produced randomly, it allows the entire range of possible solutions (the search space). Sometimes, the solutions may be "seeded" in areas where optimal solutions are likely to be found.
- **Selection**: During each successive generation, a population of the existing population is selected to create a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the latter process may be very time-consuming.
- **Reproduction**: The next step is to breed a second generation population of solutions from those selected through genetic. For each new solution to be created, a pair of "parent" solutions is selected for generating from the pool selected previously. By breeding a "child" solution using the above methods of crossover and mutation, a new solution is produced which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new generation of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more similar to nature of biology, some research suggests more than two "parents" are better to be reproduce a good quality child [13].

These processes eventually result in the next generation population of offspring that is different from the initial generation. Generally speaking the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for generating, along with a small proportion of less fit solutions, for reasons already mentioned above.

Crossover and mutation are the most famous genetic operators but it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms.

- **Termination**: This generational process is repeated until a termination condition has been reached. Common terminating conditions are:
- A solution is found that satisfies minimum criteria.
- Fixed number of generations reached.
- Allocated budget (computation time /money) reached.
- The highest level solutions fitness is reaching or has reached a plateau such that successive iterations no longer produce better results.
- Manual inspection.
- Combinations of the above.

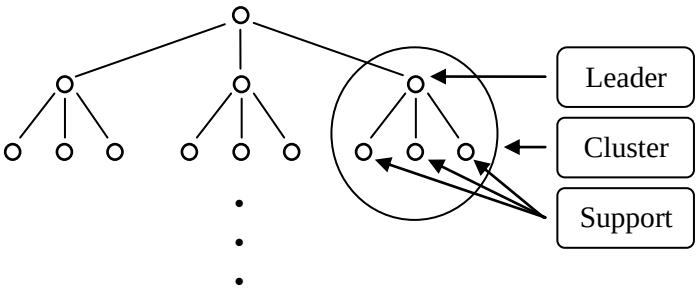
Simple generational genetic algorithm procedure:

- Choose the initial population of individuals.
- Evaluate the fitness of each individual in that population.
- Repeat on this generation until termination (time limit, sufficient fitness achieved, etc.)
- Select the best-fit individuals for reproduction.
- Generating new individuals through crossover and mutation operations to give birth to offspring.

- Evaluate the individual fitness of new individuals.
- Replace least-fit population with new individuals.

3.3 Memetic Algorithm Approach

The memetic algorithms [14] can be viewed as a marriage between a population-based global technique and a local search made by each of the individuals. They are a special kind of genetic algorithms with a local hill climbing. Like genetic algorithms, memetic algorithms are a population-based approach. They have shown that they are orders of magnitude faster than traditional genetic algorithms for some problem domains. In a memetic algorithm a population structure approach based on a ternary tree was chosen. In contrast with a non-structured population it divides the individuals in clusters and restricts crossover possibilities.



Population structure

The structure consists of several clusters and each cluster consists of a leader and three supporter solutions. The leader is chosen as the best individual of the cluster. The number of individuals in the population is defined by a number of nodes in the ternary tree, i.e., it is necessary 13 individuals to make a ternary tree with 3 levels, 40 individuals to 4 levels and so on.

- **Representation:** For the permutation flowshop scheduling problem the representation we have chosen is quite intuitive, with a solution being represented as a chromosome with the alleles assuming different integer values in the [1, *n*] interval, where *n* is the number of jobs.
- **Crossover:** As indicated in section 3.2 is the same crossovers in GA.
- **Mutation:** As indicated in section 3.2 is the same mutation in GA.
- **Fitness Function:** As in this problem the goal is to minimize the weighted mean completion time and weighted mean tardiness, the fitness function was chosen as randomly.
- **Offspring Insertion in 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 leader, the new individual takes its place. Otherwise it takes the place of the supporter that took part in the recombination. If the new individual is already present in the population, it is not inserted. We adopted a policy of not allowing duplicated individuals to reduce loss of diversity. After all individuals were inserted, the population is restructured. The fitness of the leader of a group must be lower than the fitness of the leader of the group just above it. Following this policy, the higher subgroups will have leaders with better fitness than the lower groups and the best solution will be the leader of the root subgroup. The adjustment is made by comparing the leader of each subgroup with the leader of the subgroup just above. If the leader in the level below turns out to be better, they swap their places.

4 Computational Experience

4.1 Test Problems

In this section a number of experiments are carried out which outlines the effectiveness of both the algorithm described above. The purpose of these experiments is to compare the performance of memetic algorithm approach with genetic algorithm approach for the Permutation Flowshop

Scheduling Problem. The experiments were conducted on Pentium IV at 2.2GHz, 2GB computer using 'Matlab' language.

A set of test problem was created to compare the performance of the algorithms. The main characteristics of a problem are its size as measured by the number of machines, and the number of jobs. And the degree of correlation in the processing times for each jobs are likely to effect the efficiency of algorithm which find near optimal solutions. A sample of test problems was generated with three machines and 10, 20, 30, 40, 50, 75, 100, 150, 200, 500, 1000 and 2000 jobs. This method of processing time p_{i1}, p_{i2} and p_{i3} in the test problems were randomly sampled from a uniform distribution on the integers defined on [1,10] and the due dates where generated from uniform distribution $[(1-TF-RDD/2)SP, (1-TF+RDD/2)SP]$ such that $SP=\sum_{i=1}^{n} k_i$ where $k_i=(p_{i1}+p_{i2}+p_{i3})/3$, TF = 0.2, 0.4, RDD = 0.2, 0.4, 0.6, 0.8, 1, and the due date generation follow that given in [7] and integer weights w_i and tardiness penalty h_i are drawn from distribution in the range $[1,10]$. For each value of n jobs we have average 10 problems.

4.2 Comparative Results

In this section we will report on the results of our computational tests to show the effectiveness of our local search methods. We are going to compare the results which we have pure versions of pure genetic algorithm (GA_{pur}) and pure memetic algorithm (MA_{pur}). In table (1) we compare the efficiency GA_{pur} and MA_{pur} have been approached in terms of comparable average of value (V_{avg}) and average of time (T_{avg}) in case of without using the *VNS*. The MA_{pur} is the best in case of values test problem but for the time; GA_{pur} and MA_{pur} almost the same.

Table (1) Compare between GA_{pur} and MA_{pur} for $F3 \parallel \sum_{i=1}^{n}(w_i C_{i,m}/W + h_i T_{i,m}/H)$ problem without using VNS

	GA_{pur}		MA_{pur}	
n	V_{avg}	T_{avg}	V_{avg}	T_{avg}
10	51.84193	0.124905	51.09262	0.124704
20	75.99547	0.176598	73.85434	0.176486
30	115.078	0.235044	109.0408	0.237384
40	144.3009	0.293423	141.016	0.293873
50	176.0374	0.349932	171.1646	0.353946
75	247.7387	0.517809	243.9867	0.503806
100	334.5184	0.68767	327.6341	0.677696
150	486.9344	1.043307	476.5341	1.028668
200	632.6442	1.451246	624.7556	1.479306
500	1524.405	4.989597	1516.281	5.015494
1000	3006.417	13.87517	2983.928	13.99025
2000	5939.021	47.15903	5902.931	42.60472

n: number of jobs.
T_{avg}: average time in seq.
V_{avg}: average value of the objective function.

In the following table (2) shows the efficiency local search heuristic methods (Genetic algorithm (GA_{HA}) and Memetic algorithm (MA_{HA})) have been approached in terms of comparable rate of value and time with hybrid versions with *VNS*. The *VNS* is taking from the algorithm *HA*. The GA_{HA} is the almost best value for jobs since they are goods except (10, 20, 30, 2000) jobs and also for the time; The GA_{HA} is better than the MA_{HA}.

Table (2) Compare between GA_{HA} and MA_{HA} for $F3 \parallel \sum_{i=1}^{n}(w_i C_{i,m}/W + h_i T_{i,m}/H)$ problem with using *VNS*

	GA_{HA}		MA_{HA}	
n	V_{avg}	T_{avg}	V_{avg}	T_{avg}
10	48.20147	0.128141	48.04437	0.12578
20	68.89549	0.183678	68.35036	0.180783
30	102.7427	0.237143	101.7414	0.236342
40	130.9945	0.296141	130.1191	0.31677
50	156.6596	0.349918	157.9709	0.409337
75	226.4671	0.505848	226.9081	0.527959
100	306.4692	0.759616	306.6347	0.667848
150	447.3711	1.020531	448.7808	1.029536
200	589.7393	1.470041	593.1156	1.445019
500	1455.018	4.992128	1459.727	4.939721
1000	2916.17	14.13807	2917.381	13.91558
2000	5827.962	43.6308	5825.517	42.66788

GA_{HA}: hybrid genetic algorithm.
MA_{HA}: hybrid memetic algorithm.

5 Conclusion

This paper has developed a number of solution procedures for three machines flowshop scheduling minimizing $\sum_{i=1}^{n}(w_i C_{i,m}/W + h_i T_{i,m}/H)$.

- For the pure algorithms show that pure memetic algorithm performs better than the pure genetic algorithms for all results.
- The hybrid genetic algorithms versions with *VNS*, gives good solutions better than hybrid memetic algorithm and both were better than pure algorithms.
- The local search methods used to solve all the large problems, the results show the robustness and flexibility of local search heuristics.

Future work Some suggestions for future research are described as follows:

- First, the extensions propose of the exact for $F3 \parallel \sum_{i=1}^{n}(w_i C_{i,m}/W + h_i T_{i,m}/H)$ 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] Solimanpur M., Vrat P. and Shankar R., (2004): A neuro-tabu search heuristic for flowshop scheduling problem. Computers & Operations Research 31:2151–2164.

[2] Pinedo M., (1995): Scheduling: theory algorithms and systems. Englewood Cliffs, Prentice-Hall, New Jersey.

[3] Marett R. and Wright M., (1996): A comparison of neighborhood search techniques for multi-objective combinatorial problems. Computers & Operations Research 23:465–483.

[4] Sayin S. and Karabati S., (1999): A bicriteria approach to the two-machine flowshop scheduling problem. European Journal of Operational Research 113:435–449.

[5] Danneberg D., Tautenhahn T. and Werner F., (1999): A comparison of heuristic algorithms for flowshop scheduling problems with setup times and limited batch size. Math Comput Model 29:101–126.

[6] Toktas B., Azizoglu M. and Koksalan S. K., (2004): Two-machine flowshop scheduling with two criteria: maximum earliness and makespan. European Journal of Operational Research 157:286–295.

[7] Cheachan H. A., Mohammed H. A. and Khtan Q. A., (2010): Scheduling flowshop machines to minimize the multi-objective functions. Iraqi Journal for Administrative Sciences 637–657.

[8] Ravindran D., Noorul Haq A., Selvakuar S. J. and Sivaraman R., (2005): Flowshop scheduling with multiple objective of minimizing makespan and total flow time. Int J Adv Manufacturing Tech 25:1007–1012.

[9] Loukil T., Teghem J. and Tuyttens D., (2005): Solving multi-objective production scheduling problems using metaheuristics. European Journal of Operational Research 161:42–61.

[10] Mohammed H. A., Cheachan H. A. and Khtan Q. A., (2009): Single machine scheduling to minimizing sum penalty number of late jobs subject to minimize the sum weight of completion time. Journal of Kerbala University. 7(1):163–173.

[11] Yousefi M. and Yusuff R. M., (2012): Minimizing earliness and tardiness penalties in a single machine scheduling against common due date using genetic algorithm. Research Journal of Applied Sciences, Engineering and Technology 4(9): 1205-1210.

[12] Mohammed H. A., Hassan A. S., Saloomi M. H. and Khtan Q. A., (2012): Memetic Algorithm and Genetic Algorithm for the Single Machine Scheduling Problem with Linear Earliness and Quadratic Tardiness Costs. Journal of Kerbala University. 7(1):163–173.

[13] Ting, Chuan-Kang (2005): On Mean Convergence Time of Multi-parent Genetic Algorithms without Selection. Advances in Artificial Life, pp: 403-412. ISBN 978-3-540-28848.

[14] Murata T., Ishibuchi H. and Tanaka H., (1996): Multi-objective genetic algorithm and its applications to flowshop scheduling. Computers and Industrial Engineering 30:957–968.