Applied Mathematics and Computation 205 (2008) 550–561

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Genetic algorithm integrated with artificial chromosomes for multi-objective flowshop scheduling problems

Pei-Chann Chang [a,*], Shih-Hsin Chen [b], Chin-Yuan Fan [b], Chien-Lung Chan [a]

[a] Department of Information Management, Yuan-Ze University, Chung-Li, Taiwan
[b] Department of Industrial Engineering and Management, Yuan-Ze University, Chung-Li, Taiwan

ARTICLE INFO

Keywords:
Genetic algorithm
Flowshop scheduling
Makespan
Maximum tardiness
NSGA II

ABSTRACT

Recently, a wealthy of research works has been dedicated to the design of effective and efficient genetic algorithms in dealing with multi-objective scheduling problems. In this paper, an artificial chromosome generating mechanism is designed to reserve patterns of genes in elite chromosomes and to find possible better solutions. The artificial chromosome generating mechanism is embedded in simple genetic algorithm (SGA) and the non-dominated sorting genetic algorithm (NSGA-II) to solve single-objective and multi-objective flowshop-scheduling problems, respectively. The single-objective problems are to minimize the makespan while the multi-objective scheduling problems are to minimize the makespan and the maximum tardiness. Extensive numerical studies are conducted and the results indicate that artificial chromosomes embedded with SGA and NSGAII are able to further speed up the convergence of the genetic algorithm and improve the solution quality. This promising result may be of interests to industrial practitioners and academic researchers in the field of evolutionary algorithm or machine scheduling.

1. Introduction

In the operations research literature, flowshop scheduling is one of the most well studied problems in the area of scheduling. Garey et al. [14] prove that the permutation flowshop scheduling problem is NP-complete. Moreover, if the flowshop scheduling problem with multiple objectives is considered, it becomes even more complicated. Therefore, researchers start to develop effective heuristics and meta-heuristics to solve this problem. Among the meta-heuristics, genetic algorithms attract a lot of attention. Jaszkiewicz [18], Ishibuchi et al. [17], Reeves [28], Murata et al. [24], and Chen et al. [10] do the pioneering work by applying genetic algorithms in solving flowshop scheduling problems. Reeves and Yamada [27], Wang and Zhang [29], Wang et al. [31], Chang et al. [9], hybridize other techniques with genetic algorithms to improve the genetic search. Numerous multi-objective algorithms are also proposed, such as segregative genetic algorithms by Affenzeller [1] which combine populations when the diversity of a population was decreased; multisexual genetic algorithm by Lis and Eiben [22] which assigns each chromosome with different sex and it restricts only different sex that can be mated; Hierarchical Fair Competition Model by Hu et al. [16] which divides the population into a hierarchical structure; genetic local search by Jaszkiewicz [18] which uses a weighted sum of multi-objective as fitness and randomly sets the weight values while two parent solutions are selected. This genetic local search shows better performance in a reasonable computational time; MO Particle Swam Optimization by Coello et al. [12] and Nojima et al. [25]; two-phase subpopulation GA by Chang

* Corresponding author.
E-mail address: iepchang@saturn.yzu.edu.tw (P.-C. Chang).

0096-3003/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2008.05.027

et al. [6] which simultaneously applies several subpopulation and assigns the weight for these subpopulation to explore the solution space uniformly; Mining Gene subpopulation GA by Chang et al. [5] which employs a mining gene technique based on the subpopulation genetic algorithm; and multi-objective Tchebycheff based genetic algorithm by Alves and Almeida [2] which proposes a similar idea to the two-phase subpopulation genetic algorithm [6]. Finally, the latest approach is to apply the Immune algorithm to solve multi-objective problems [36].

NSGA-II in Deb et al. [13] and SPEA2 in Zitzler et al. [32] are two well-known algorithms in solving multi-objective optimization problems. The main characteristic of NSGA-II is to sort the chromosomes into different ranks. During the selection procedure, while there are two solutions from the same rank, a crowding distance is applied to determine which one is to be selected. As for the fitness assignment of SPEA2, it is based on the number of dominated solutions from Pareto set and from each chromosome. For a complete review, please refer to Jones et al. [19]. Therefore, different EMO algorithms are proposed with the goal to further improve the efficiency and solution quality of the algorithm. In recent years, evolutionary algorithm with probability models (EAPM) is the most important branch of GA. The reason is that the problem-independent crossover operator may either break the building block of chromosomes or does not mix the genetic information properly [33]. Thus, EAPM employs the probability model to generate new chromosomes instead of using crossover or mutation operator.

In this research, we take a close look at the evolutionary process for a permutation flowshop scheduling problems and come out with the new idea of generating artificial chromosomes to further improve the solution quality of the genetic algorithm. To generate artificial chromosomes, it depends on the probability of each job at a certain position. The idea is originated from Chang et al. [7,8] which propose a methodology to improve GAs by mining gene structures within a set of elite chromosomes generated in previous generations. Instead of replacing the crossover operator and mutation operator due to efficiency concern, the proposed algorithm is embedded into SGA and NSGA-II. The probability model acquired from the elite chromosomes will be integrated with the genetic operators in generating artificial chromosomes, i.e., offsprings which can be applied to enhance the efficiency of the proposed algorithm. Apart from our previous researches, Harik et al. [15], Rastegar and Hariri [26], Zhang et al. [30] have discussed and proved the genetic algorithm which is based on the probability models. For a complete review of the relative algorithms discussed above, please refer to Larrañaga and Lozano [21], Lozano et al. [23], and Pelikan et al. [33]. In most recent works of EAPM, they all concentrate on solving continue problems rather than discrete problems. There are only few researches [7,30,34,38–40] in applying EAPM to resolve discrete problems.

In summary, this research proposes an artificial chromosome generating mechanism (AC), which is embedded in simple genetic algorithm (SGA) and in non-dominated sorting genetic algorithm II (NSGA-II), which are called ACGA and ACNSGA-II in solving the single-objective and multi-objective flowshop scheduling problems, respectively. By examining the evolutionary process, a fitness based gene probability matrix is developed to guide the searching procedure. In addition, the proposed approach is also applied to illustrate how insights gained which can be further converted into our understanding of EA's behaviors in developing new and better techniques. The proposed algorithm will be tested on flowshop scheduling problem in minimizing makespan for single-objective problems and in minimizing makespan and maximum tardiness for the multi-objective problems. The study adopts the flowshop scheduling instances provided by Reeves [28] and Ishibuchi et al. [17] for benchmark tests.

The rest of the research is organized as follows: Section 2 gives the problem statement, and Section 3 describes the methodology of generating artificial chromosomes. Section 4 explains the detail procedures of the proposed algorithm. In Section 5, extensive experiments are conducted to test the performance of the proposed algorithm in single-objective and multi-objective scheduling problems. Finally, the conclusion is discussed and future researches are also provided.

2. Problem statement

Flowshops are useful tools in modeling manufacturing processes. A permutation flowshop is a job processing facility, which consists of several machines and several jobs to be processed on the machines. In a permutation flowshop all jobs follow the same machine or processing order. Our objectives are to find a set of compromise solutions so that the makespan and maximum tardiness are minimized.

The flowshop scheduling problem is a typical assembly line problem where n different jobs have to be processed on m different machines. All jobs are processed on all the machines in the same order. The processing times of the jobs on machines are fixed irrespective of the order in which the processing is done. The problem is characterized by a matrix $P = (p_{ij})$, $i = 1 \ldots n$, $j = 1 \ldots m$, of processing times. Each machine processes exactly one job at a time and each job is processed on exactly one machine at a time. The problem then is to find a sequence of jobs such that the makespan that is the completion time of the last job in the sequence on the last machine is minimized. If C_i denotes the completion time for job i, then we are trying to minimize max C_i. There are many other criterions that can be considered for optimization. We refer the reader to Bagchi [3] for a detailed discussion of multi-objective scheduling using GA. For details of the flowshop and other scheduling and sequencing problems we refer the reader to Baker [4].

The flow shop scheduling can be formerly defined as follows: if $p(i,j)$is the processing times for job i on machine j, and a job permutation $\{\pi_1,\pi_2, \ldots ,\pi_n\}$, where there are n jobs and m machines, then the completion times $C(\pi_i, j)$ is calculated as follows:

$$\begin{aligned} &C(\pi_1,1) = p(\pi_1,1)\\ &C(\pi_i,1) = C(\pi_{i-1},1) + p(\pi_i,1) \quad \text{for} \quad i = 2,\ldots,n\\ &C(\pi_1,1) = C(\pi_1,j-1) + p(\pi_1,j) \quad \text{for} \quad j = 2,\ldots,m\\ &C(\pi_i,j) = \max\{C(\pi_{i-1},j), C(\pi_i,j-1)\} + p(\pi_i,j) \quad \text{for} \quad i = 2,\ldots,n; \quad j = 2,\ldots,m. \end{aligned} \tag{1}$$

The makespan is finally defined as

$$C_{\max}(\pi) = C(\pi_n, m). \tag{2}$$

Then, the objective is to find a permutation π^* in the set of all permutations Π such that

$$C_{\max}(\pi) \leqslant C_{\max}(\pi) \quad \forall \pi \in \Pi. \tag{3}$$

A more general flowshop scheduling problem can be defined by allowing the permutation of jobs to be different on each machine. However, what work has been done to show on the more general flow shop scheduling problem has tended to small improvement in solution quality over the permutation flowshop scheduling problems (PFSP) while increasing the complexity of the problem substantially. The size of the solution space increases from $n!$ to $(n!)^m$. Other objective functions for the PFSP also received a lot of attentions. For example, the mean flow-time (the time a job spends in process), or the mean tardiness (assuming some deadline for each job) are to be minimized. Other real problems from the manufacturing industries such as jobs may have non-identical release dates, there may be sequence-dependent setup times, and there may be limited buffer storage between machines and so on. These characteristics of the real world problems will make the problem more complicated to be solved within a reasonable time frame. However, GA approaches provide a more realistic view to the problem. Since it can generate alternatives of sequences (in the evolving process each chromosome representing a feasible solution to the problem) to the decision maker, a more applicable sequence can be decided to solve the current problem with satisfactory results.

3. Generating artificial chromosomes

During the evolving process of the GA, all the chromosomes will converge slowly into certain distribution after the final runs. If we take a close look at the distribution of each gene in each assigned position, we will find out that most the genes will be converged into certain locations which means the gene can be allocated to the position if there is a probabilistic matrix to guide the assignment of each gene to each position.

Artificial Chromosomes are developed according to this observation and a dominance matrix will record this gene distribution information. The dominance matrix is transformed into a probability matrix to decide the next assignment of a gene to a position. Consequently, AC is integrated into the procedure of genetic algorithm and it attends to improve the performance of genetic algorithm. The primary procedure is to collect gene information first and to use the gene information to generate artificial chromosomes. Before collecting the gene information, AC collects the chromosomes whose fitness is better by comparing the fitness value of each chromosome with average fitness value of current population. Then artificial chromosome is embedded into the genetic algorithm. The detailed steps are described in the following:

Step 1: To convert gene information into dominance matrix: Before we collect gene information, selection procedure is performed to select a set of chromosomes. Then, for a selected chromosome, if job i exists at position j, the frequency is added by 1. To demonstrate the working theory of the artificial chromosome generation procedure, a 5-job problem is illustrated. Suppose there are ten sequences (chromosomes) whose fitness is better than average fitness. Then, we accumulate the gene information from these ten chromosomes to form a dominance matrix. As shown in the left-hand side of Fig. 1, there are two job 1, two job 2, 2 two 3, one job 4, and three job 5 on position 1. Again, there are 3 job 1, 1 job2, 2 job3, 3 job4, and 1 job5 on position 2. The procedure will repeat for the rest of the position. Finally, the dominance matrix contains the gene information from better chromosomes is illustrated in the right-hand side of Fig. 1.

Step 2: Generate artificial chromosomes: As soon as we collect gene information into dominance matrix, we are going to assign jobs onto the positions of each artificial chromosome. The assignment sequence for every position is assigned randomly, which is able to diversify the artificial chromosomes. After we determine the assignment sequence, we select one job assigned to each position by roulette wheel selection method based on the probability of each job on this position. After we assign one job to a position, the job and position in the dominance matrix are removed. Then, the procedure continues to select the next job until all jobs are assigned. Assume the first job is to be assigned at position 3 in the beginning, which is shown in Fig. 2. The frequency of each job at position 3 is [1, 3, 1, 1, and 4] starting from job 1 to job 5. Because the number of total frequency is 10, the corresponding probability for job 1 is 1/10; job 2 is 3/10, and so on. Then, we accumulate the probability from job 1 to 5 and roulette wheel select is able to apply this accumulated probability. This information is shown at the last column of the Fig. 2. If a random probability 0.6 is generated, then job 4 is assigned to position 3.

Step 3: Replacement strategy: After embedding artificial chromosomes into the population, we use $\mu + \lambda$ strategy, which combines previous parent population and artificial chromosomes. Then, we select better μ chromosomes from the combined population. Consequently, better solutions are preserved to the next generation.

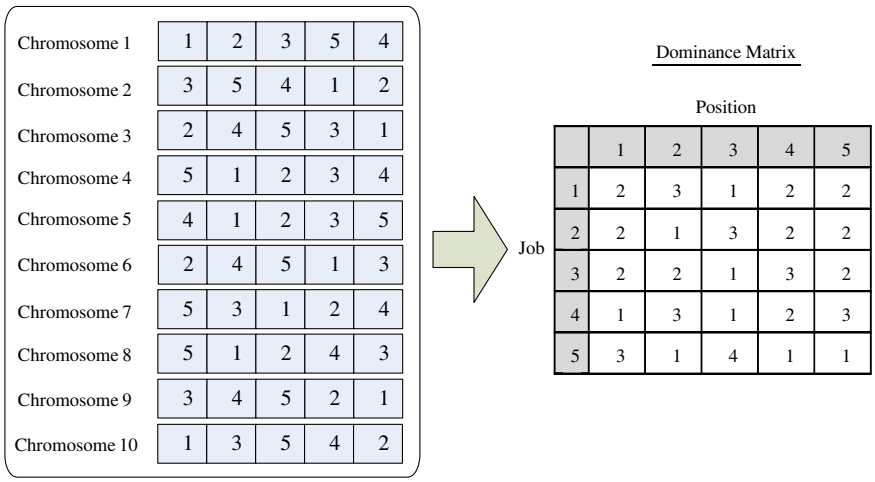

Fig. 1. To collect gene information and converted into a dominance matrix.

Job	Pos. 3	Prob.	Accum
1	1	1/10	1/10
2	3	3/10	4/10
3	1	1/10	5/10
4	1	1/10	6/10
5	4	4/10	10/10

Fig. 2. The probability and accumulated probability of each job for position 3.

During the assignment of each job to a specific position, the dominance matrix will be updated continuously. For example, after assigning job 4 at position 3 and suppose position 2 is the next one to be assigned. An updated dominance matrix is shown in Fig. 3.

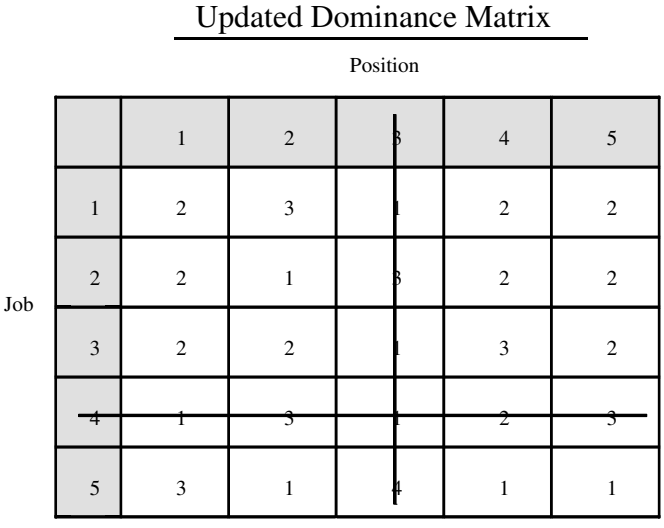

Fig. 3. The updated dominance matrix after assigning job 4 at position 3.

Job	Pos. 2	Prob .	Accum
1	3	3/7	3/7
2	1	1/7	4/7
3	2	2/7	6/7
5	1	1/7	7/7

Fig. 4. The probability and accumulated probability of each job for position 2.

Next, the probability of each job is recalculated as well as the accumulated probability as shown in Fig. 4. Then, a roulette wheel selection method will select a job based on the probability of each job. Consequently, the algorithm iteratively assigns jobs to vacant positions until all jobs are assigned.

4. Implementations of AC with genetic algorithm and NSGA-II

AC is embedded with SGA and NSGA-II algorithms to further improve the solution quality of these two algorithms. However, in order to provide a good quality of artificial chromosomes the timing to collect this gene information will be very crucial. Since during the early stage of evolutionary process, the chromosomes generated may not yet converge well into a certain quality, it is not a good idea to collect gene information at this stage. The collecting procedure may have to hold until some generations after. Therefore, the first parameter *startingGen* determines the time to collect gene information and generate AC. Secondly, AC is not generated every generation because it is very time-consuming; normally AC is generated under a pre-determined interval, ex., every 50 generations. This is to make sure that the procedure will not converge prematurely and to save computational times.

The next section explains the integration of AC with SGA. Section 4.2 introduces the NSGA II, and Section 4.3 describes the procedures of the ACNSGA-II algorithm. Finally, in order to verify the solution quality obtained by the proposed algorithm, there are two performance metrics employed to evaluate the solution quality, which are explained in Section 4.4.

4.1. Embedding AC with SGA

The artificial chromosome operator has two parameters, i.e., *startingGen* and generation *interval*, to be setup. The detail procedures of the artificial chromosome operator are described as follows:

MainProcedure

Population: The population used in the genetic algorithm
Generations: The number of generations
startingGen: It determines when to collect gene information and to generate AC
interval: The generation interval to generate artificial chromosomes

```
1. Initiate Population
2. counter ← 0
3. while counter < generations do
4.        Evaluate Objective and Fitness()
5.        FindEliteSolutions(i)
6.        if counter < startingGen or counter %      Interval! = 0 do
7.        Selection with Elitism Strategy()
8.       Crossover()
9.        Mutation()
10.        TotalReplacement()
11.              else
12.                    CalculateAverageFitness()
13.                    CollectGeneInformation()
14.                    GenerateArtificialChromsomomes()
15.                    μ + λ Replacement()
16.              End if
17.              counter ← counter + 1
18.   end while
```

The CollectGeneInformation() procedure is described at the step 1 of Section 3, which collects gene information from better chromosomes depended on the average fitness of the population. Generate-artificial-chromsomomes() follows the step 2 of Section 3, which uses the probability selection to assign job onto each positions. Finally, after we evaluate the fitness of artificial chromosomes, the parent chromosomes and artificial chromosomes are combined into together, whose population size is $\mu + \lambda$. Then, we select the size of μ from the $\mu + \lambda$ chromosomes deterministically. The new population becomes the parent chromosomes and the proposed algorithm employs it to continually evolve. Consequently, better solutions are preserved to the next generation.

4.2. Introduction of NSGA-II

Among all the multi-objective algorithms, the major difference is the fitness assignment. The fitness assignment of NSGA-II [13] is depended on the Pareto dominance of ranking value. The lower the ranking level, the better solution quality is. When two solutions are selected from the same rank, NSGA-II adopts a crowding distance to measure the density of individuals in solution space. The following remarks NSGA-II that can be divided in three parts:

1. Non-dominated sorting: N populations and their N subpopulations first compose of a $2N$ population and then they are sorted according to each individual's domination situation.
2. Crowding distance computation: Crowding distance computation is used to decrease the competitive ability of the non-dominated solutions with more crowding distance.
3. Crowded computation operator $\prec$n: This operator is used as a selection tool. The selection is based on two comparison rules: (1) the smaller level the individual belongs to, the better the solution is; (2) an individual with greater crowding distance has better solution because the area it belongs to is less crowded.

Because the concept of NSGA-II is simple, it is the most well-known multi-objective algorithm. The next section explained how AC works with NSGA-II.

4.3. Embedding AC with NSGA-II

After the introduction of NSGA-II, this section describes how the AC is embedded with NSGA-II. The main steps of artificial chromosome embedded in NSGA-II are explained as follows:

MainProcedure

Population: The population used in the genetic algorithm
Generations: The number of generations
startingGen: It determines when does the AC works
interval: The frequency to generate artificial chromosomes

```
1. Initiate Population
2. ConstructInitialPopulation(Population)
3. counter ← 0
4. while counter < generations do
5.         Evaluate Objectives ()
6.         FindEliteSolutions(i)
7.         if counter < startingGen or counter % interval != 0 do
8.                 Perform NSGA II()
9.         else
10.                 Selection()
11.                 CollectGeneInformation()
12.                 GenerateArtificialChromsomomes()
13.                 μ + λ Replacement()
14.        End if
15.        counter ← counter + 1
16. end while
```

The only differences are that the selection operator and the source of collecting gene information. In the single-objective problem, binary tournament selection is employed and NSGA-II applies the non-dominant ranking and crowded distance. In addition, in order to keep the diversity gene information in multi-objective problems, all the chromosomes are used to build the probability model while only better chromosomes are employed in building the probability model.

Finally, because the flowshop scheduling problem with minimizing makespan (Z_{TC}) and maximum tardiness (Z_{TT}) are to be dealt with in this paper, the objective function "Evaluate Objectives ()" will calculate these two objective values for the corresponding chromosomes or solutions. The equations of these two objectives are defined below.

$$Z_{TC} = \max\{F_1, F_2, \ldots, F_m\}, \tag{5}$$

$$Z_{TT} = \max\{T_1, T_2, \ldots, T_m\}, \tag{6}$$

where F_i is the makespan on each machine and $T_i = \max\{C_i - d_i, 0\}$

4.4. Performance metric

The research adopts $D1_R$ and C metric to evaluate the solution quality for the multi-objective problems. $D1_R$ is a metric, which considers the convergence and diversity at the same time according to Knowles and Corne [20]. After a test run, the algorithm will obtain a set of Pareto solutions, which are compared with a reference set. Thus, $D1_R$ value is obtained. The lower $D1_R$ value is, the better the solution quality is. Therefore, the $D1_R$ provides a basis for comparing the performance among different algorithms in the study. The equation of $D1_R$ is listed as follows:

$$D1_R(A_j) = \frac{1}{|Z^*|} \sum_{y \in Z^*} \min\{d_{xy} | x \in A_j\}, \tag{7}$$

$$d_{xy} = \sqrt{\sum_{i=1}^{n} (f_i^*(y) - f_i(y))^2}, \tag{8}$$

where A_j is a set of Pareto solution obtained by an algorithm; Z^* is the reference solution or true Pareto solution; $|Z^*|$ is the number of reference solution.

C metric is applied to compare two solution sets A and B by transforming the relationship (A, B). If $C(A, B)$ is 1, then all the solutions of algorithm B is covered by A set and vice versa. This evaluation value is between 0 and 1. The equation of C metric is listed as follows:

$$C(A, B) \triangleq \frac{|\{b \in B | \exists a \in A, a \preceq b\}|}{|B|_c} \tag{9}$$

5. Experimental tests

This section presents the experiments of combining AC with GA in single objective problems and NSGA-II in multi-objective problems, which are named ACGA and AC-NSGAII, respectively. Standard benchmark instances of single-objective flowshop problems are obtained from http://mscmga.ms.ic.ac.uk by Reeves [28], which minimizes the makespan. When we solved the multi-objective problems, the instances are provided by Ishibuchi[1] et al. [17] where the first objective is makespan and the second objective is maximum tardiness. Both of them are shown in the following sections:

5.1. Single-objective flowshop problems

Simple genetic algorithm (SGA) and ACGA are used to solve the 21 problems named Rec01, Rec03–Rec41. Both of them are compared with hybrid genetic algorithm (HGA) from literature [41] which is the case of infinite buffer. The total number of examined solutions is n^*m^* 50 where n is the number of jobs and m is the number of machines. The population size, crossover rate, and mutation rate of SGA and ACGA are the same, which are 100, 0.9, and 0.5, respectively. There are two parameters of ACGA, which are *startingGen* and *interval*. The first parameter "*startingGen*" represents that artificial chromosome generation mechanism starts to activate from the "Starting Generation". The second parameter "*interval*" denotes how long the artificial chromosome generation mechanism has to wait to collect higher quality chromosomes. Table 1 is the parameter configuration of ACGA from design of experiments.

The ANOVA Table indicates that factor interval is significant. As shown in Figs. 5 and 6, the starting generation of ACGA is set up as 3/10 of the total generations and every 1/10 of the total generations are set up as the interval.

As soon as the parameters of ACGA are set up, the average performance measures of ACGA and SGA are shown in Table 2. As shown in Table 2, ACGA outperforms SGA in every instance which shows the superiority of AC in further speeding up the convergence of the global searching capability of GA. The average error ratios of these three algorithms including HGA are summarized in Table 3. As shown in Table 3, HGA only performs better than SGA in larger size instances such as Rec31–Rec41. However, ACGA outperforms SGA and HGA in all cases. Thus, ACGA is superior to SGA and HGA in the single-objective problems.

[1] http://www.ie.osakafu-u.ac.jp/~hisaoi/ci_lab_e/index.html.

Table 1
Design of experiments for ACGA

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Instance	20	4E+09	4E+09	2E+08	468,102	**0.000**
StartingGen	1	22	22	22	0.05	0.822
Interval	1	2951	2951	2951	6.74	**0.009**
Instance * startingGen	20	18,109	18,109	905	2.07	**0.004**
Instance * interval	20	6463	6463	323	0.74	0.790
StartingGen * interval	1	47	47	47	0.11	0.744
Instance * startingGen * interval	20	16,575	16,575	829	1.89	**0.010**
Error	2436	1E+06	1E+06	438		
Total	2519	4E+09				

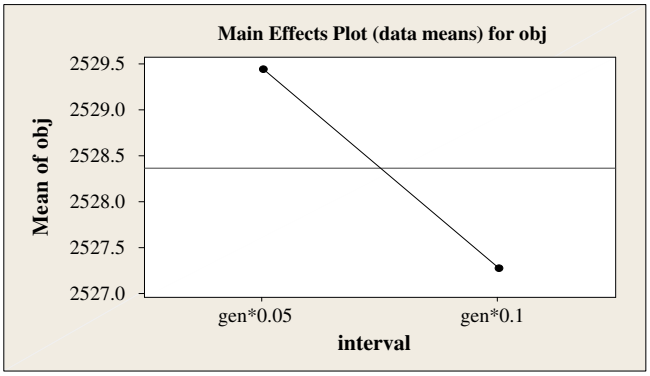

Fig. 5. Main effect plot of interval of ACGA.

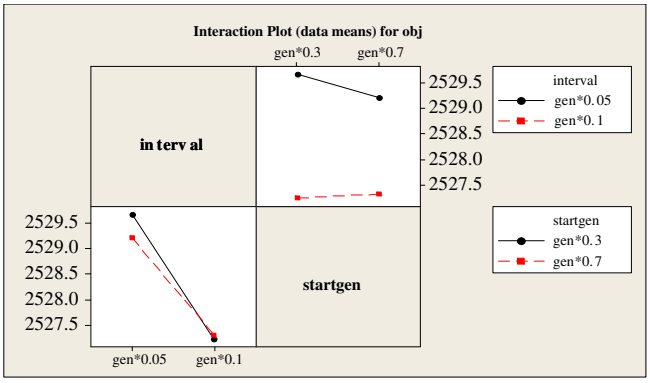

Fig. 6. Interaction plot of interval of ACGA and the starting generation.

5.2. Multi-objective flowshop problems

This section discusses experimental results of the proposed method which is embedded in NSGA-II in solving the multi-objective flowshop problems. There are four instances whose job sizes are 20, 40, 60, and 80 jobs and each of them contains 20 machines in the flowshop. The stopping criterion is to examine 100, 000 solutions totally, i.e., to generate 100,000 chromosomes. In ACNSGA-II, two parameters, i.e., *startingGen* and *interval*, have to be set up before hand. Consequently, a parameter configuration experiment by design of experiment is implemented. The following ANOVA Table indicates that there is no significant difference in different levels of these two parameters. As a result, *startingGen* and *intervals* are set up as 500 and 40, respectively according to the design of experiments for AC-NSGAII as shown in Table 4.

According to the design of experiment by Chen [11], the parameters' settings of NSGA-II in crossover rate, mutation rate, and population size are set up to 1.0, 1.0, and 100, respectively. The proposed algorithm, i.e., ACNSGA-II, is compared with NSGA-II using D1r and C metrics. Table 5 compares the performance of NSGA-II and ACNSGA-II in *D1r* metric and Fig. 7 provides results of these two methods in *C* metric.

Table 2
Average objective values of SGA and ACGA in solving standard benchmark

Instance	n, m	Opt	SGA			ACGA		
			Min	Mean	Max	Min	Mean	Max
rec01	20, 5	1247	1249	1252.2	1280	1249	1249	1249
rec03	20, 5	1109	1109	1112.3	1117	1109	1110.9	1116
rec05	20, 5	1242	1245	1251.2	1273	1245	1245.6	1262
rec07	20, 10	1566	1584	1586.8	1626	1566	1578	1584
rec09	20, 10	1537	1538	1568.7	1589	1537	1552.5	1574
rec11	20, 10	1431	1431	1445.4	1476	1431	1438.6	1469
rec13	20, 15	1930	1936	1959.8	1981	1935	1951.3	1981
rec15	20, 15	1950	1961	1980.5	2011	1950	1966.9	1995
rec17	20, 15	1902	1919	1957.4	2009	1911	1938.5	1961
rec19	30, 10	2093	2124	2152.3	2197	2099	2130.9	2182
rec21	30, 10	2017	2050	2063.3	2103	2046	2052.2	2081
rec23	30, 10	2011	2041	2070.6	2104	2021	2047.9	2079
rec25	30, 15	2513	2554	2595.5	2660	2545	2576.7	2629
rec27	30, 15	2373	2402	2438.9	2484	2396	2422.9	2468
rec29	30, 15	2287	2324	2367.2	2423	2304	2349.7	2412
rec31	50, 10	3045	3124	3185.3	3264	3105	3155.7	3261
rec33	50, 10	3114	3140	3180.9	3231	3140	3150.2	3173
rec35	50, 10	3277	3277	3308.9	3370	3277	3281.5	3291
rec37	75, 20	4951	5210	5274.7	5351	5193	5254.6	5340
rec39	75, 20	5087	5266	5338.4	5442	5276	5338.7	5427
rec41	75, 20	4960	5215	5289.4	5356	5208	5279.8	5342

Table 3
Average error ratios of these three algorithms (%)

Instance	SGA	ACGA	HGA	Instance	SGA	ACGA	HGA
rec01	0.42	0.16	1.36	rec23	2.96	1.83	3.47
rec03	0.3	0.17	1.35	rec25	3.28	2.53	3.69
rec05	0.74	0.29	0.4	rec27	2.78	2.1	2.8
rec07	1.33	0.76	1.91	rec29	3.51	2.74	3.92
rec09	2.06	1.01	2.26	rec31	4.61	3.64	3.88
rec11	1.01	0.53	3.09	rec33	2.15	1.16	2.08
rec13	1.54	1.1	2.08	rec35	0.97	0.14	0.21
rec15	1.56	0.87	1.66	rec37	6.54	6.13	4.77
rec17	2.91	1.92	3.36	rec39	4.94	4.95	3.62
rec19	2.83	1.81	2.85	rec41	6.64	6.45	5.53
rec21	2.3	1.74	2.5	Overall	2.64	2	2.7

Table 4
Design of experiments for AC-NSGAII

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Size	3	1E+07	1E+07	3E+06	793.79	0
Starting	1	550	550	550	0.13	0.723
Interval	1	4700	4700	4700	1.08	0.3
Size * Starting	3	10240	10240	3413	0.78	0.505
Size * Interval	3	7551	7551	2517	0.58	0.631
Starting * Interval	1	203	203	203	0.05	0.829
Size * Starting * Interval	3	5898	5898	1966	0.45	0.717
Error	464	2E+06	2E+06	4368		
Total	479	1E+07				

Table 5
The comparison of NSGA-II and ACNSGA-II

Size	NSGA-II			ACNSGA-II		
	Min	Mean	Max	Min	Mean	Max
20	23.33	43.05	81.74	25.2	45.23	86.46
40	101.95	145.03	252.22	92.67	156.9	212.24
60	217.4	334.3	452	222.1	320.3	473.6
80	242.3	425	682.7	238.9	422.7	711.6

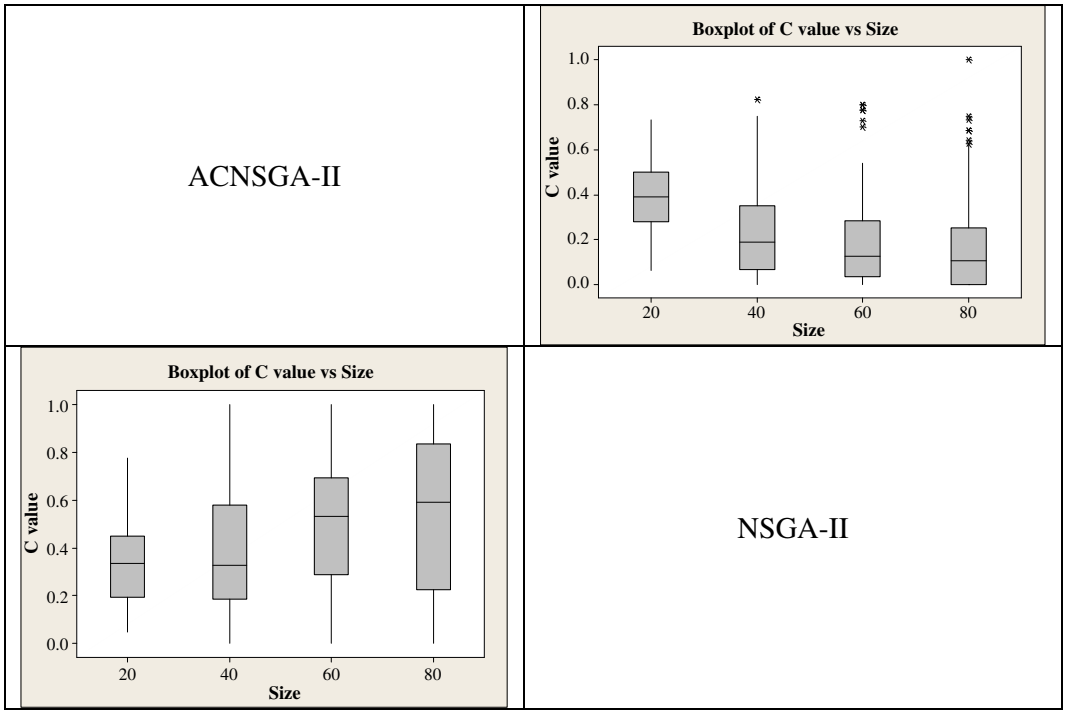

Fig. 7. The C metric performance of ACNSGA-II and NSGA-II under different job sizes.

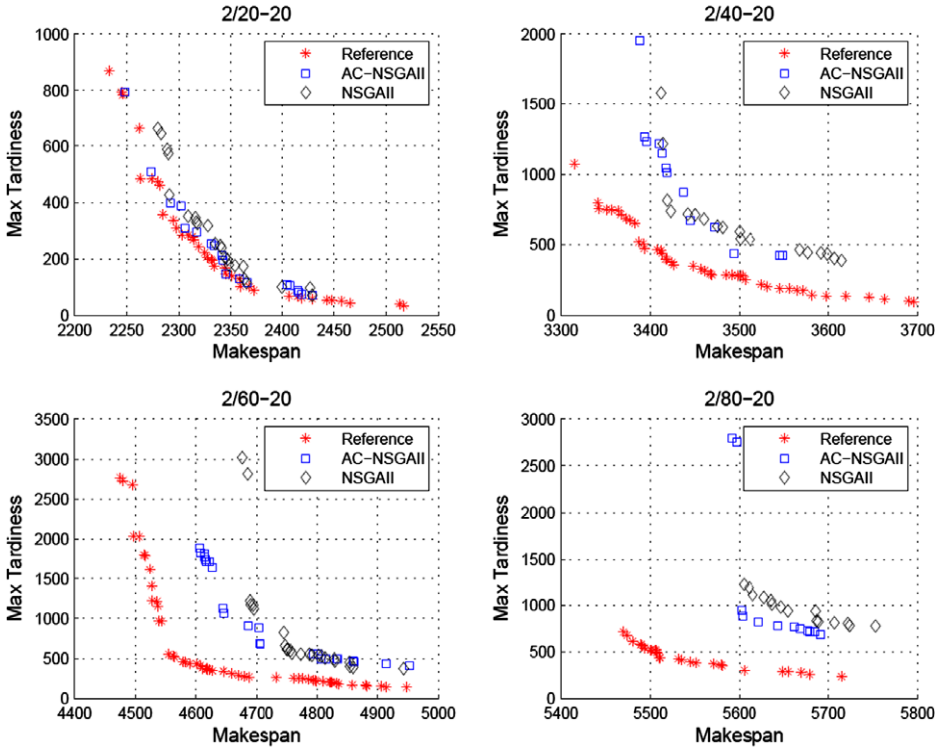

Fig. 8. A single run of the non-dominated solutions by NSGA-II and AC-NSGA-II.

In Table 5, NSGA-II performs a little bit better for small size problems while ACNSGA-II is much better in dealing these large size problems. In other words, ACNSGA-II is more effective in speeding up the convergence when the problem size is large. The minimum value in D1r metric provided by these two methods also indicates that ACNSGA-II is quite effective in further improving the solution quality of the problems. Besides, *C* metric shows ACNSGA-II covers more solutions of NSGA-II in job size 60 and 80. However, there is no difference in smaller instance, including job 20 and job 40. Consequently, AC-NSGAII is able to perform well in larger size problem from these numerical results. Finally, Fig. 8 is the non-dominated solutions of a single run by NSGA-II and AC-NSGA-II. The results show that the proposed algorithm works effectively in large size problems.

6. Conclusions and future researches

In this paper, an artificial chromosome generating mechanism is proposed, which can be classified in the class of Evolutionary Algorithm with Probability Models. Previous researches attempt to replace crossover operator and mutation operators. However, this research embedded the gene based probability models in SGA and NSGA-II and the experimental results show that the proposed mechanism is promising in dealing with flowshop scheduling problems in single-objective and multi-objective problems. In our experience, this hybrid method does not have a heavy burden in computational times and the proposed method functions more efficiently than EAPM without using crossover and mutation operator. In the near future, properties and special characteristics of generating artificial chromosomes will be further investigated. They are still open problems in how and when to generate effective artificial chromosomes in speeding up the convergence process while still maintaining the global searching capability. It is also another interesting subject as to generate diversified artificial chromosomes in improving the diversity of the population. These diversified AC can help GA in searching or exploring different solution spaces thus providing possibility in discovering a solution with better quality. In addition, the gene based probability model applied in generating AC is able to determine the mutation rate as in [35] and to design competent crossover and mutation operators as in [37]. Finally, different applications of the AC generating mechanism are also possible in continuous and other combinatorial optimization problems.

References

[1] M. Affenzeller, New Generic Hybrids Based Upon Genetic Algorithms, Institute of Systems, Science Systems Theory and Information Technology, Johannes Kepler University, 2001.
[2] M.J. Alves, M. Almeida, MOTGA, A multiobjective Tchebycheff based genetic algorithm for the multidimensional knapsack problem, Comput. Oper. Res. 34 (11) (2007) 3458–3470.
[3] T. P Bagchi, Multiobjective Scheduling by Genetic Algorithms, Kluwer academic publishers., 1999.
[4] K.R. Baker, Introduction to Sequencing and Scheduling, Wiley, New York, 1974.
[5] P.C. Chang, S.H. Chen, C.H. Liu, Sub-population genetic algorithm with mining gene structures for flow shop scheduling problems, Expert Syst. Appl. 33 (3) (2007) 762–771.
[6] P.C. Chang, S.H. Chen, K.L. Lin, Two-phase sub population genetic algorithm for parallel machine-scheduling problem, Expert Syst. Appl. 29 (3) (2005) 705–712.
[7] P.C. Chang, S.H. Chen, C.Y. Fan, Mining gene structures to inject artificial chromosomes for genetic algorithm in single machine scheduling problems, Appl. Soft. Comput. 8 (1) (2008) 767–777.
[8] P.C. Chang, Y.W. Wang, C.H. Liu, in: New Operators for Faster Convergence and Better Solution Quality in Modified Genetic Algorithm, LNCS 3611 (2005) 983–991.
[9] P.C. Chang, J.C. Hsieh, C.H. Liu, A case-injected genetic algorithm for single machine scheduling problems with release times, Int. J. Prod. Econ. 103 (2006) 551–564.
[10] C.L. Chen, V.S. Vempati, N. Aljaber, An application of genetic algorithms for flow shop problems, Eur. J. Oper. Res. 80 (1995) 389–396.
[11] Chi-Chia Chen, Mining Gene Structures with Inheritance Sub-Population Genetic Algorithm in Solving Combinatorial Problem, Department of Industrial Engineering and Management, Master thesis, University of Y.Z., Taiwan, 2006.
[12] C.A.C. Coello, G.T. Pulido, M.S. Lechuga, Handling multiple objectives with particle swarm optimization, IEEE Trans. Evol. Comput. 8 (3) (2004) 256–279.
[13] K. Deb, S.A. Amrit Pratap, T. Meyarivan, A fast and elitist multi objective genetic algorithm-NSGA-II, in: Proceedings of the Parallel Problem Solving from Nature VI Conference, 2000, pp. 849–858.
[14] M.R. Garey, D.S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Math. Oper. Res. 1 (1976) 117–129.
[15] G.R. Harik, F.G. Lobo, D.E. Goldberg, The compact genetic algorithm, IEEE Trans. Evol. Comput. 3 (4) (1999) 287–297.
[16] J.E. Hu, K. Goodman, Z. Fan Seo, R. Rosenberg, The hierarchical fair competition framework for sustainable evolutionary algorithms, Evol. Comput. 13 (2) (2005) 241–277.
[17] H. Ishibuchi, T. Yoshida, T. Murata, Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling, IEEE Trans. Evol. Comput. 7 (2) (2003) 204–223.
[18] A. Jaszkiewicz, Genetic local search for multi-objective combinatorial optimization, Eur. J. Oper. Res. 137 (1) (2002) 50–71.
[19] D.F. Jones, S.K. Mirrazavi, M. Tamiz, Multi-objective meta-heuristics: an overview of the current state-of-the-art, Eur. J. Oper. Res. 137 (1) (2002) 1–9.
[20] J.D. Knowles, D.W. Corne, On metrics for comparing non dominated sets, in: Proceedings of the 2002 Congress on Evolutionary Computation Conference (CEC02), IEEE Press, New York, 2002, pp. 711–716.
[21] P. Larrañaga, J.A. Lozano, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer, Norwell, MA, 2001.
[22] J. Lis, A.E. Eiben, A multisexual genetic algorithm for multicriteria optimization, in: Proceedings of the 4th IEEE Conference on Evolutionary Computation, 1997, pp. 59–64.
[23] J.A. Lozano, P. Larrañaga, I. Inza, E. Bengoetxea, Towards a New Evolutionary Computation, Springer, 2006.
[24] T. Murata, H. Ishibuchi, H. Tanaka, Genetic algorithms for flowshop scheduling problems, Comput. Ind. Eng. 30 (1996) 1061–1071.
[25] Y. Nojima, K. Narukawa, S. Kaige, H. Ishibuchi, Effects of removing overlapping solutions on the performance of the NSGA-II algorithm, in: Proceedings of the 3rd International Conference on Evolutionary Multi-Criterion Optimization, 2005, pp. 341–354.
[26] R. Rastegar, A. Hariri, A step forward in studying the compact genetic algorithm, Evol. Comput. 14 (3) (2006) 277–289.
[27] C.R. Reeves, T. Yamada, Genetic algorithms, path relinking, and the flowshop sequencing problem, Evol. Comput. 6 (1998) 45–60.
[28] C.R. Reeves, A genetic algorithm for flowshop sequencing, Comput. Oper. Res. 22 (1995) 5–13.

[29] L. Wang, D.Z. Zheng, An effective hybrid heuristic for flow shop scheduling, Int. J. Adv. Manuf. Technol. 21 (2003) 38–44.
[30] Q. Zhang, J. Sun, E. Tsang, An evolutionary algorithm with guided mutation for the maximum clique problem, IEEE Trans. Evol. Comput. 9 (2) (2005) 192–200.
[31] L. Wang, L. Zhang, D.Z. Zheng, An effective hybrid genetic algorithm for flow shop scheduling with limited buffers, Comput. Oper. Res. 33 (2006) 2960–2971.
[32] E. Zitzler, M. Laumanns, S. Bleuler, A tutorial on evolutionary multi objective optimization, in: Proceedings of The Workshop on Multiple Objective Metaheuristics, 2004.
[33] M. Pelikan, D.E. Goldberg, F.G. Lobo, A survey of optimization by building and using probabilistic models, Comput. Optim. Appl. 21 (1) (2002) 5–20.
[34] U. Aickelin, E.K. Burke, J. Li, An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering, J. Oper. Res. Soc. 58 (12) (2007) 1574–1585.
[35] H. Mühlenbein, L. Zinchenko, V. Kureichik, T. Mahnig, Effective mutation rate for probabilistic evolutionary design of analogue electrical circuits, Appl. Soft. Comput. 7 (3) (2007) 1012–1018.
[36] K.C. Tan, C.K. Goh, A.A. Mamun, E.Z. Ei, An evolutionary artificial immune system for multi-objective optimization, Eur. J. Oper. Res. 187 (2) (2008) 371–392.
[37] C.F. Lima, D.E. Goldberg, K. Sastry, F.G. Lobo, Combining competent crossover and mutation operators: a probabilistic model building approach, in: Proceedings of the 2005 conference on Genetic and evolutionary computation, Washington, DC, USA, 2005, pp. 735–742.
[38] P.A.N. Bosman, D. Thierens, Permutation optimization by iterated estimation of random keys marginal product factorizations, Parallel Problem Solving from Nature – PPSN VII 331-340, 2002.
[39] S. Tsutsui, M. Pelikan, A. Ghosh, Edge histogram based sampling with local search for solving permutation problems, Int. J. Hyb. Inform. Sys. 3 (1) (2006) 11–22.
[40] Q. Zhang, J. Sun, E. Tsang, Combinations of estimation of distribution algorithms and other techniques, Int. J. Aut. Comput 4 (3) (2007) 273–280.
[41] L. Wang, L. Zhang, D.Z. Zheng, An effective hybrid genetic algorithm for flow shop scheduling with limited buffers, Comput. Oper. Res. 33 (2006) 2960–2971.