AN APPLICATION OF EFFECTIVE GENETIC ALGORITHMS FOR SOLVING HYBRID FLOW SHOP SCHEDULING PROBLEMS

CENGIZ KAHRAMAN*
Department of Industrial Engineering, İstanbul Technical University, Maçka, İstanbul, Turkey.

ORHAN ENGIN
Department of Industrial Engineering, Selçuk University, Konya, Turkey.

İHSAN KAYA
Department of Industrial Engineering, İstanbul Technical University, Maçka, İstanbul, Turkey.

MUSTAFA KERIM YILMAZ
Department of Industrial Engineering, Selçuk University, Konya, Turkey.

Received:04-09-2007 Revised:04-12-2007

This paper addresses the Hybrid Flow Shop (HFS) scheduling problems to minimize the makespan value. In recent years, much attention is given to heuristic and search techniques. Genetic algorithms (GAs) are also known as efficient heuristic and search techniques. This paper proposes an efficient genetic algorithm for hybrid flow shop scheduling problems. The proposed algorithm is tested by Carlier and Neron's (2000) benchmark problem from the literature. The computational results indicate that the proposed efficient genetic algorithm approach is effective in terms of reduced total completion time or makespan (C_{max}) for HFS problems.

Key words: Hybrid flow shop scheduling, Genetic algorithm, completion time

1. Introduction

A HFS scheduling problem consists of series of production stages, each of which has several machines operating in parallel. Some stages may have only one machine, but at least one stage must have multiple machines. Each job is processed by one machine in each stage and it must go through one or more stages. Machines in each stage can be identical, uniform or unrelated (Linn and Zhang, 1999). The hybrid flow shop scheduling problems can be formally described as follows (Engin and Döyen 2004): Machines are arranged into s stages in series; in each stage k $(k=1,....,s)$ there are m_k identical machines in parallel; job j, $j=1,....,n$, has to be processed on any machine at each stage and job j has finite processing times in each stage $(p_{1j}, p_{2j},...., p_{sj})$. We assume that all jobs and machines are always available during the scheduled period and the preemption is not allowed. The objective is to find a schedule which minimizes the maximum completion time (makespan). HFS problems are NP-Hard when the objective is to minimize the makespan (Gupta, 1988).

A hybrid flow shop scheduling problem's mathematical model is described as a mix integer programming. The model is given as follows (Hong Wang 1998):

The notation of the HFS model;

J : The set of the jobs to be scheduled, $|J| = n$,

s : The set of the stages that all the jobs will be processed, $|s| = s$,

j : Subscript representing job j,

l : Subscript representing stage l,

* Corresponding Author:
E-Mail: kahramanc@itu.edu.tr
Phone: +90-212-296 40 40
Fax:+90-212-240 72 60

International Journal of Computational Intelligence Systems, Vol.1, No. 2 (May, 2008), 134-147

i : Subscript representing machine *i*,

m_l : The number of the machines at stage *l*,

B: a very large positive number,

S_{jl} : Starting time of job *j* at stage *l*,

$\forall j, j \in J, \forall l, l \in s$,

$$X_{jli} = \begin{cases} 1, & \text{if job } j \text{ is on machine } i \text{ at stage } l. \\ 0, & \text{otherwise} \end{cases}; \quad (1)$$
$$\forall j, j \in J, \forall l, l \in s, i = 1, \ldots\ldots, m_l$$

$$Y_{fgli} = \begin{cases} 1, & \text{if job } f \text{ is before job } g \text{ on machine } i \text{ at stage } l \\ 0, & \text{otherwise} \end{cases}; \quad (2)$$
$$\forall j, j \in J, \forall l, l \in s, i = 1, \ldots\ldots, m_l$$

p_{lj}: Processing time of job *j* at stage *1*;
$\forall j, j \in J, \forall l, l \in s$,

Minimize Q

Subject to

$$S_{js} + p_{js} \le Q; \ \forall j \quad (2.1)$$

$$S_{jl} + p_{jl} \le S_{j,l+1}; \ \forall j, l, l \ne s \quad (2.2)$$

$$S_{fl} + p_{fl} \le S_{gl} + B(1 - Y_{fgli}); \ \forall l, i, f, g, f \ne g \quad (2.3)$$

$$Y_{fgli} + Y_{gfli} \le 1; \ \forall l, i, f, g, f \ne g \quad (2.4)$$

$$X_{fli} + X_{gli} \le 1 + Y_{fgli} + Y_{gfli}; \ \forall l, i, f, g, f \ne g \quad (2.5)$$

$$\sum_{i=1}^{m_l} X_{jli} = 1 \qquad \forall j, l \quad (2.6)$$

$$S_{jl} \ge 0; \qquad \forall j, l \quad (2.7)$$

$$X_{jli}, \ Y_{fgli} \in \{0,1\} \quad (2.8)$$

Constraint (2.1) indicates that the completion time of the last job at the last stage *s* is *Q*. Constraint (2.2) indicates that it is not possible for job *j* to be processed at stage *l*+1 before job *j* at stage *l* is completed. The processing order for jobs *f* and *g* on machine *i* at stage *l* is defined by constraints (2.3), (2.4), and (2.5). These constraints do not allow more than one job to be processed on a machine at any time. Constraint (2.6) does not allow a job to be processed on more than one machine at any time. Constraints (2.7) and (2.8) provide that the variables are nonnegative and 0-1 integer values.

A HFS problem with 5-jobs×3-stages where the stages have 2, 3, and 3 machines respectively is given in Fig. 1.

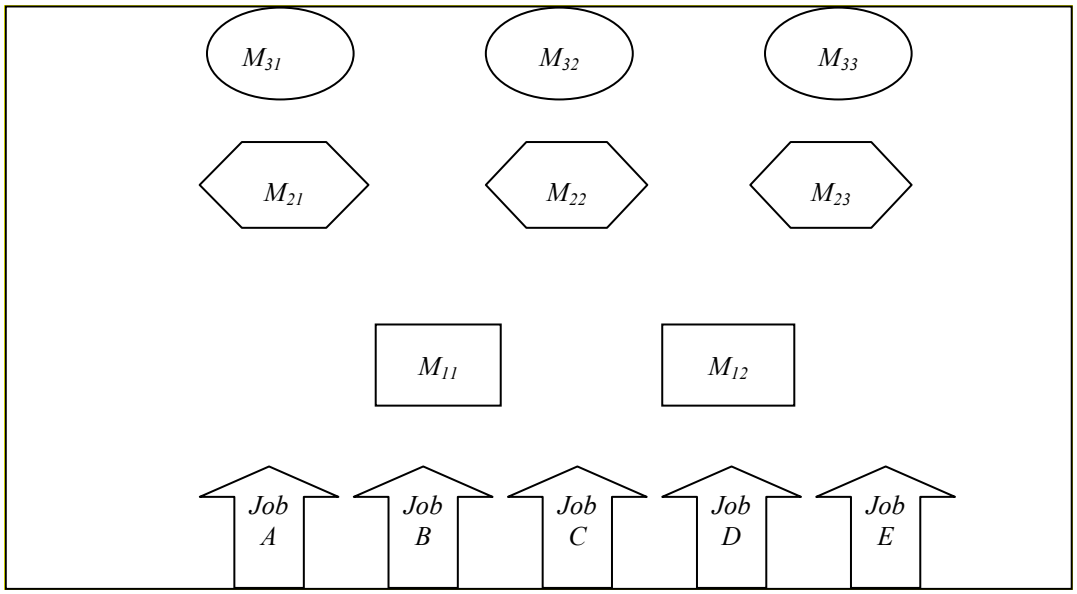

Figure 1. An Illustrative Hybrid Flow Shop Scheduling Problem

The HFS was first studied by Arthanari and Ramamurthy (1971). They developed a branch and bound algorithm for solving HFS problems. Gupta (1988) and Hoogeveen et al. (1996) proved that the two stage hybrid flow shop scheduling problem was NP-Hard in the stronge sense even if there was only one machine on the first stage and two machines on the second stage. The other studies on the hybrid flow shop scheduling problems in the literature are given in Table 1.

Table 1. Literature review on HFS scheduling problems.

The Authors	Year	Problem	Algorithm
Brah and Hunsucker	1991	Flow shop with multiple processors	Branch and bound
Portmann et al.	1998	HFS	Genetic Algorithm
Riane et al.	1998	HFS	Heuristic methods
Jessen and Weizhen	1998	HFS	On line algorithm
Moursli and Pochet	2000	HFS	Branch and bound
Soewandi and Elmaghraby	2001	Flexible flow shops	Heuristic method
Neron et al.	2001	HFS	Branch and bound
Engin and Döyen	2004	HFS	Artificial immune system
Yang et al.	2005	Complex- HFS	Heuristic method
Tang et al.	2005	Dynamic-HFS	Neural network
Zhong et al.	2006	Multi-objective- HFS	Evolutionary algorithm
Haouari et al.	2006	Two stage- HFS	Branch and bound
Allaoui and Artiba	2006	Two stage- HFS	Branch and bound
Zandieh et al.	2006	HFS	Immune algorithm
Janiak et al.	2007	HFS	Constructive and metaheuristics algorithms
Vob and Witt	2007	HFS	Heuristic solution
Caricato et al.	2007	HFS	Heuristic method
Alaykıran et al.	2007	HFS	Ant colony optimization

International Journal of Computational Intelligence Systems, Vol.1, No. 2 (May, 2008), 134-147

There are a few studies on hybrid flow shop scheduling problems solved by GAs.

Wang et al. (2006) proposed an effective hybrid genetic algorithm (HGA) for permutation flow shop scheduling with limited buffers. In the HGA, not only multiple genetic operators based on evolutionary mechanism are used simultaneously in hybrid sense, but also a neighborhood structure based on graph model is employed to enhance the local search, so that the exploration and exploitation abilities can be well balanced. They investigated the effects of buffer size and decision probability on optimization performances using simulation.

Oğuz and Ercan (2005) proposed a GA for hybrid flow-shop scheduling with multiprocessor task problems and described its implementation. They improved a new crossover operator to be used in the GAs and compared it with Partially Matched Crossover (PMX). They also employed a preliminary test to establish the best combination of the control parameters to be used along with different genetic operators.

Li-Xin et al. (2002) developed a genetic descent algorithm for hybrid flow shop scheduling problem. Randomly generated 230 instances were tested by simulation program. Computational experiments show that for small size HFSS scheduling problems, the average deviation of GDA from the optimal solution is 0.01%; for medium-large size problems, the performance of GDA is 10.45% better than that of NEH algorithm.

Xia et al. (2000) proposed a GA approach for hybrid flow shop scheduling problem. The algorithm is based on the list scheduling principle by developing job sequences for the first stage and queuing the remaining stages in a FIFO manner.

In this paper, an effective GA is developed for HFS scheduling problems. The effectiveness of the proposed method is tested with Carlier and Neron's (2000) HFS scheduling problems from the literature. The computational results indicate that the proposed approach is effective in terms of reduced makespan for the attempted problems. To the best of our knowledge, there are no genetic algorithms applied to Hybrid flow shop including Carlier and Neron's (2000) scheduling problems in the literature.

The rest of the paper is organized as follows. The proposed effective algorithm is explained in Section 2. In Section 3, an extensive computational study using the proposed algorithm and experiments are presented. In Section 4, the paper is concluded with some comments.

2. Genetic Algorithms

GAs were invented by John Holland (Goldberg, 1989) and they were stochastic search methods designed to search large and complex spaces by exploitation of currently known solutions and a robust exploration of the entire search space (Yoon and Ventura, 2002).

GAs use a collection of solutions called population. Each individual in the population is called a chromosome (a string of symbols) and a chromosome represents a solution to the problem. The chromosomes can be produced through successive iterations, called generations and the population size (the number of individuals in a population) remains fixed from generation to generation. The chromosomes are evaluated using the value of the fitness function during each generation. A set of genetic operators such as reproduction (selection) and recombination (crossover and mutation) is applied to create new and better solutions (off springs) from the individuals of the current population and the solutions are steadily improved from generation to generation. The structure of GAs is given in Fig. 2.

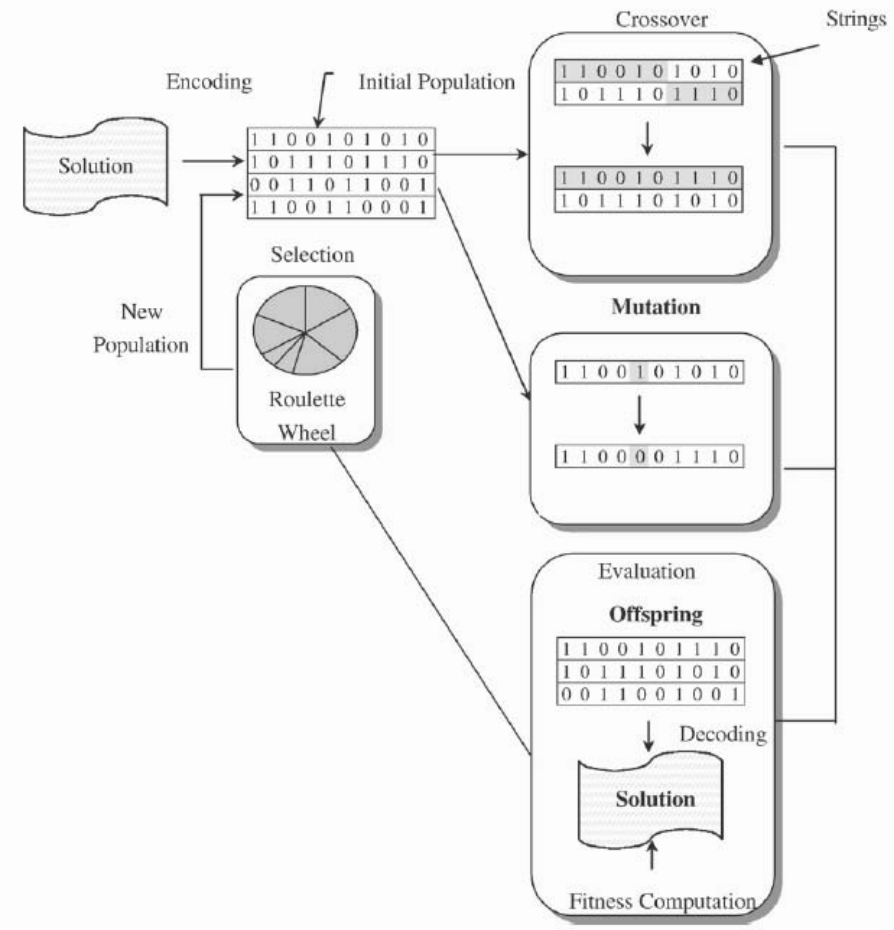

Figure 2. The fundamental cycle and operations of basic GAs (Gen and Cheng, 2000)

The proposed GA is based on a permutation representation of the n jobs. The details of our implementation for the GAs are given as follows.

A direct coding approach is used. In this coding, a chromosome represents a schedule directly (Yamada et al., 1992). The initial population is randomly generated. The population size is determined by the help of a full factorial experimental design using our GA program.

Selection schemes allow the algorithm to take biased decisions favoring good strings when generations change. For this aim, some of the good strings are replicated while some of bad strings are removed. As a consequence, after the selection mechanism is determined, the population is likely to be "dominated" by good strings. Various selection schemes in the literature have been used. We focus on roulette wheel selection and tournament selection without replacement.

The fitness function plays an important role in deciding the string in the next generation. The fitness function of a string is defined by the makespan (C_{max}) value of the schedule.

Crossover is used as the main genetic operator and the performance of a GA is heavily dependent on it. During the past three decades, various crossover operators have been proposed for the scheduling problems. In this study, six crossover operators have been used: Position Based Crossover (*PBX*), Order Crossover (*OX*), Partially Mapped Crossover (*PMX*), Cycle Crossover (*CX*), Linear Order Crossover (*LOX*) and Order Based Crossover (*OBX*) that are widely used in the literature. These six crossover operators are briefly explained in the following:

PBX

First, it is generated a random mask and then exchanged relative genes between parents according to the mask. This operator is explained and detailed in section 3.1.

International Journal of Computational Intelligence Systems, Vol.1, No. 2 (May, 2008), 134-147

OX

The offspring inherits the elements between the two crossover points from the selected parent in the same order and position as they appear in the parent. The remaining elements are inherited from the alternate parent in the order in which they appear in that parent, beginning with the first position following the second crossover point and skipping over all elements already present in the off spring (Kaya and Engin, 2007; Cheng et al. 1999).

PMX

A parent and two crossover sites are selected randomly and the elements between two string positions in one of the parents are directly inherited by the offspring. Each element between the two crossovers points in the alternate parent are mapped to the position held by this element in the first parent. Then the remaining elements are inherited from the alternate parent (Kaya and Engin, 2007; Cheng et al. 1999).

CX

The cycle between strings is fined, the symbols in the cycle are coped to a new string, the remaining symbols are determined for the new string by deleting the symbols and the remaining symbols are fulfilled with the new string.

LOX

The two sublists are selected from strings randomly. $Sublist_2$ is removed from $string_1$, leaving some "holes" and then holes are slide from the extremities toward the center until they reach the cross section. Similarly, $Sublist_1$ is removed from $string_2$. At the end, $Sublist_1$ is inserted into the holes of $string_2$ to form $offspring_1$ and $sublist_2$ is inserted into the holes of $string_1$ to form $offspring_2$ (Gen and Cheng, 2000).

OBX

A set of positions is selected randomly; the order of symbols in the selected positions is imposed on the corresponding symbols in the other string.

Mutation operator plays a very important role in GAs and it helps maintain diversity in the population to prevent premature convergence. Six mutation operators are examined in the GA to minimize the makespan in HFS. These are neighborhood based, adjacent two job change, arbitrary two job change, arbitrary three job change, shift change and inversion mutation operator.

3. Computational Results

The proposed GA can be summarized as follows;

```
Set GAs value:
        Size of initial population:
        Choose selection method:
        Choose crossover method:
        Choose mutation method:
        Set selection, crossover and mutation ratio:
        Set generation size:
        Set CPU time:
End
For initial population
                Evaluate chromosome by randomly;
                Evaluate makespan value;
                End if
        Next:
        Evaluate selection ratio
        Do
                Choose genes for selection according to the ratio
                Add it initial population
                Eliminate others:
        Evaluate chromosome ratio
        Do
                Choose genes for crossover according to ratio
                Crossover;
                Evaluate makespan value
                End if
        Loop until reach crossover ratio
        Sort the chromosomes in ascending order depending on makespan value:
        Select chromosomes as many as initial population sizes:
        Do
                Choose two genes for mutation according to ratio;
                Mutation;
                End if
        Loop until reach mutation ratio
        While stopping criteria= false:
```

3.1. *Parameter optimization for GAs*

It is well known that GAs' efficiency depends on a high degree upon the selection of the control parameters. GAs' search process is controlled with multiple factors (control parameters) whose effects will possibly interact with each other. In general, there are a few control mechanisms for these parameters and in this paper the full factorial Design of Experiments (DOE) is used. The application involves six parameters (factors), each having possible different values. These parameters are given in Table 2.

Table 2. The levels of GA control parameters

Control Parameters	Levels
Selection methods	Roulette wheel, Tournament.
Selection ratios	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
Crossover Methods	PBX, OX, PMX, CX, LOX, OBX
Crossover Ratios	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0.
Mutation Methods	Neighborhood based, adjacent two job change, arbitrary two job change, arbitrary three job change, shift change, and inversion
Mutation Ratios	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0

The benchmark problems given in Carlier and Neron (2000) are considered in the study. The problem size varies from 10 job×5 stages to 15 job×10 stages. Processing times have a uniform distribution in the range of (3, 20). Three characteristics that define a problem are *no. of jobs*, *no. of stages* and *no. of identical machines* at each stage. Total 77 problems are classified into 13

International Journal of Computational Intelligence Systems, Vol.1, No. 2 (May, 2008), 134-147

groups according to their characteristics. An instance problem is taken from each of the groups. Parameter optimization is implemented and the best parameter set is found for the instance. The parameter set found for an instance is generalized and used for the other problems in the same group. Therefore, parameter optimization is implemented for 13 instances. In the study, two selection methods, ten selection ratio levels, six crossover methods, ten crossover ratio levels, five mutation methods, and ten mutation ratio levels are implemented among the 13 problems. A total number of *2x10x6x10x6x10 = 72 000* runs are made among these problems. The best parameters set in each of the replicated runs for 13 benchmark problems are given in Table 3. The initial population is selected as 25 for all benchmark problems.

Table 3. The best parameters sets of 13 benchmark problems

Problem	Selection Method	Selection Ratio	Crossover Method	Crossover Ratio	Mutation Method	Mutation Ratio
j10c5a2	Roulette wheel	0.4	Position Based Crossover	0.3	Inversion mutation	0.1
j10c5b1		0.4		0.3		0.1
j10c5c1		0.4		0.3		0.1
j10c5d1		0.4		0.3		0.1
j10c10a1		0.1		0.2		0.1
j10c10b1		0.1		0.2		0.1
j10c10c1		0.1		0.2		0.1
j15c5a1		0.1		0.3		0.2
j15c5b1		0.1		0.3		0.2
j15c5c1		0.1		0.3		0.2
j15c5d1		0.1		0.3		0.2
j15c10a1		0.2		0.1		0.1
j15c10b1		0.2		0.1		0.1

The best selection, crossover and mutation methods for 13 benchmark problems are briefly described as follows:

Roulette wheel selection

Roulette wheel selection is chosen, where the average fitness of each chromosome is calculated depending on the total fitness of the whole population. The chromosomes are randomly selected proportional to their average fitness. Roulette wheel selection is summarized in the following steps,

Step1. Let the *pop-size*, number of strings in *pop*.

Step2. *nsum*, sum of all of the fitness values of the strings in *pop*; form *nsum* slots and assign string to the slots according to the fitness value of the string.

Step3. Do step 4 (*pop-size* -1) times.

Step4. Generate a random number between 1 and *nsum*, and use it to index into the slots to find the corresponding string; add this string to *newpop*

Step5. Add the string with the highest fitness value in *pop* to *newpop*.

Position Based Crossover (PBX)

(i) Select a set of positions from one string at random,
(ii) Produce a new string by copying the symbols on these positions into the corresponding positions in the new string,
(iii) Delete the symbols already selected from the second string. The resulting sequence contains only the symbols that the new string needs,
(iv) Place the symbols into unfixed positions in the new string from left to right according to the order of the sequence used to produce one offspring.

Inversion Mutation

It can be seen from Fig. 3. that the inversion mutation selects two positions at random and then swaps the genes on these positions.

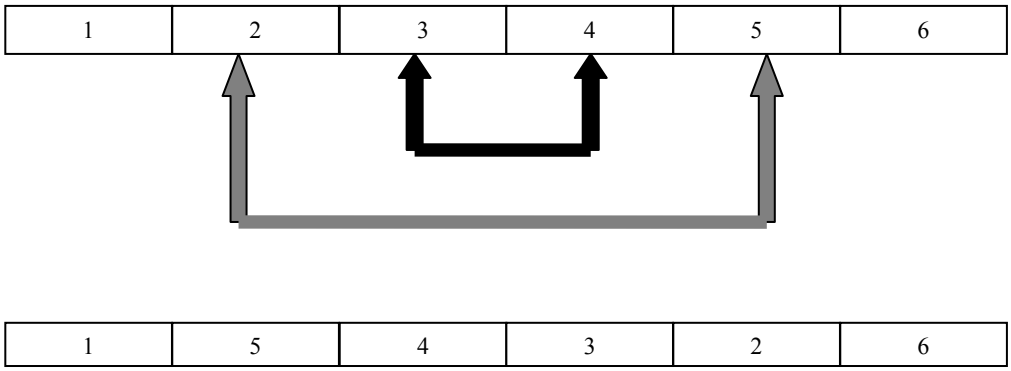

Figure 3. The Inversion mutation operators

3.2. *Computational study*

The test problems used in the experiments are Carlier and Neron's (2000) benchmark problems. The same problems were also studied by Santos et al. (1995) Engin and Döyen (2004) and Alaykiran et al. (2007). Santos et al. (1995) used a branch and bound method, Engin and Döyen (2004) used an artificial immune systems (AIS) method which was improved with the use of satisfability tests and time-bound adjustments and also Alaykiran et al. (2007) used ant colony optimization method. Santos et al. (1995), Engin and Döyen (2004) and Alaykiran et al. (2007) limited their algorithm with 1600 s. If an optimal solution was not found within 1600s, the search was stopped and the best solution was accepted as the final schedule. They calculated the Lower Bounds (*LB*) of the problems and the relative gap from these bounds for the non-optimally solved instances.

In this study, using the lower bounds, the percentage deviation from LB is calculated as

$$\%\ Deviation = \frac{Best\ C\max - Lower\ Bound\,(LB)}{Lower\ Bound\,(LB)} x100 \qquad (3)$$

The iteration number is selected as 1000 and only one replicated for all benchmark problems. Also CPU time is limited to 1600 s. If an optimal solution is not found within this time, the search is stopped and the best solution is accepted as the final schedule. The algorithm is implemented in Borland Delphi and run on a PC Pentium 4 processor with 3 GHz and 512 MB memory. In Table 4, for all of the 77 problems, the best C_{max} values and CPU times obtained by the proposed GA model, Engin and Döyen's (2004) AIS model, and Neron et al's (2001) B&B model are presented. For all methods (GA, AIS and B&B) the CPU times are given in seconds. The lower bounds and % deviations from lower bounds are given at the last three columns of Table 4.

Table 4: Solutions of Test Problems

Problem	GA Cmax	GA CPU	AIS Cmax	AIS CPU	B & B Cmax	B & B CPU	LB of Cmax	GA % deviation	AIS % deviation	B & B % deviation
j10c5a2	88	0.000	88	1	88	13	88	0	0	0
j10c5a3	117	0.000	117	1	117	7	117	0	0	0
j10c5a4	121	0.015	121	1	121	6	121	0	0	0
j10c5a5	122	0.000	122	1	122	11	122	0	0	0
j10c5a6	110	0.015	110	4	110	6	110	0	0	0
j10c5b1	130	0.000	130	1	130	13	130	0	0	0
j10c5b2	107	0.000	107	1	107	6	107	0	0	0
j10c5b3	109	0.000	109	1	109	9	109	0	0	0
j10c5b4	122	0.000	122	2	122	6	122	0	0	0
j10c5b5	153	0.000	153	1	153	6	153	0	0	0
j10c5b6	115	0.000	115	1	115	11	115	0	0	0
j10c5c1	68	0.031	68	32	68	28	68	0	0	0
j10c5c2	74	0.016	74	4	74	19	74	0	0	0
j10c5c3	**71**	**0.016**	**72**	**a**	**71**	**240**	**71**	**0**	**1.4**	**0**
j10c5c4	66	0.031	66	3	66	1017	66	0	0	0
j10c5c5	78	0.094	78	14	78	42	78	0	0	0
j10c5c6	69	0.000	69	12	69	4865b	69	0	0	0
j10c5d1	66	0.046	66	5	66	6490b	66	0	0	0
j10c5d2	73	0.110	73	31	73	2617b	73	0	0	0
j10c5d3	64	0.015	64	15	64	481	64	0	0	0
j10c5d4	70	0.000	70	5	70	393	70	0	0	0
j10c5d5	66	0.031	66	1446	66	1627b	66	0	0	0
j10c5d6	62	0.062	62	8	62	6861b	62	0	0	0
j10c10a1	139	0.015	139	1	139	41	139	0	0	0
j10c10a2	158	0.125	158	18	158	21	158	0	0	0
j10c10a3	148	0.047	148	1	148	58	148	0	0	0
j10c10a4	149	0.141	149	2	149	21	149	0	0	0
j10c10a5	148	0.000	148	1	148	36	148	0	0	0
j10c10a6	146	0.156	146	4	146	20	146	0	0	0
j10c10b1	163	0.000	163	1	163	36	163	0	0	0
j10c10b2	157	0.131	157	1	157	66	157	0	0	0
j10c10b3	169	0.000	169	1	169	19	169	0	0	0
j10c10b4	159	0.015	159	1	159	20	159	0	0	0
j10c10b5	165	0.016	165	1	165	33	165	0	0	0
j10c10b6	165	0.016	165	1	165	34	165	0	0	0
j10c10c1	115	0.062	115	a	127	c	113	1.8	1.8	12.4
j10c10c2	**117**	**0.141**	**119**	**a**	**116**	**1100**	**116**	**0.86**	**2.6**	**0**
j10c10c3	116	0.234	116	a	133	c	98	18.4	18.4	35.7
j10c10c4	120	0.281	120	a	135	c	103	16.5	16.5	31.1
j10c10c5	**125**	**0.721**	**126**	**a**	**145**	**c**	**121**	**3.3**	**4.1**	**19.8**
j10c10c6	106	0.046	106	a	112	c	97	9.3	9.3	15.5

Table 4: Solutions of Test Problems (continued)

Problem	GAs Cmax	GAs CPU	AIS Cmax	AIS CPU	B &B Cmax	B & B CPU	LB of Cmax	GAs % deviation	AIS % deviation	B & B % deviation
j15c5a1	178	0.031	178	1	178	18	178	0	0	0
j15c5a2	165	0.015	165	1	165	35	165	0	0	0
j15c5a3	130	0.015	130	1	130	34	130	0	0	0
j15c5a4	156	0.015	156	2	156	21	156	0	0	0
j15c5a5	164	0.046	164	1	164	34	164	0	0	0
j15c5a6	178	0.032	178	1	178	38	178	0	0	0
j15c5b1	170	0.015	170	1	170	16	170	0	0	0
j15c5b2	152	0.015	152	1	152	25	152	0	0	0
j15c5b3	157	0.015	157	1	157	15	157	0	0	0
j15c5b4	147	0.015	147	1	147	37	147	0	0	0
j15c5b5	166	0.016	166	2	166	20	166	0	0	0
j5c5b6	175	0.015	175	1	175	23	175	0	0	0
j15c5c1	85	0.031	85	774	85	2131b	85	0	0	0
j15c5c2	91	0.156	91	a	90	184	90	1.1	1.1	0
j15c5c3	87	0.109	87	16	87	202	87	0	0	0
j15c5c4	89	0.000	89	317	90	c	89	0	0	1.1
j15c5c5	**75**	**A**	**74**	**a**	**84**	**c**	**73**	**2.27**	**1.4**	**15.1**
j15c5c6	91	0.047	91	19	91	57	91	0	0	0
j15c5d1	167	0.015	167	1	167	24	167	0	0	0
j15c5d2	84	0.406	84	a	85	c	82	2.4	2.4	3.7
j15c5d3	83	0.015	83	a	96	c	77	7.8	7.8	24.7
j15c5d4	84	0.188	84	a	101	c	61	37.7	37.7	65.6
j15c5d5	80	0.105	80	a	97	c	67	19.4	19.4	44.8
j15c5d6	**82**	**0.406**	**82**	**a**	**87**	**c**	**79**	**2.53**	**3.8**	**10.1**
j15c10a1	236	0.015	236	1	236	40	236	0	0	0
j15c10a2	200	0.015	200	30	200	154	200	0	0	0
j15c10a3	198	0.063	198	4	198	45	198	0	0	0
j15c10a4	225	0.031	225	12	225	78	225	0	0	0
j15c10a5	182	0.016	182	2	183	c	182	0	0	0.5
j15c10a6	200	0.031	200	2	200	44	200	0	0	0
j15c10a1	222	0.031	222	3	222	70	222	0	0	0
j15c10b2	187	0.047	187	1	187	80	187	0	0	0
j15c10b3	222	0.015	222	1	222	80	222	0	0	0
j15c10b4	221	0.016	221	1	221	84	221	0	0	0
j15c10b5	200	0.094	200	1	200	84	200	0	0	0
j15c10b6	219	0.031	219	1	219	67	219	0	0	0

a: GAs and AIS could not reach LB value in 1600 s., b: B&B reaches LB value more than 1600 s , c: B&B could not reach LB value, B&B CPU

As it will be noticed from Table 4, better results for *a* and *b* type problems than *c* and *d* type problems have been obtained. The machine configurations have an important effect on the complexity of problems that effects the solution quality. GA algorithm has found the optimal

International Journal of Computational Intelligence Systems, Vol.1, No. 2 (May, 2008), 134-147

solutions for all *a* and *b* type problems like AIS algorithm (47 problems), although B&B has found the optimal solutions for 46 problems. *c* and *d* type problems are relatively hard problems. Neron et al. (2001) grouped some of the problems as *hard problems.* For these problems, they could not reach the optimal solutions in a short time. The difference of these problems is mainly sourced from their machine configurations (all of these problems are *c* or *d* type problems). There are 30 problems in that group (the *c* and *d* types of *10x5* and *15x5* problems). The rest of the problems (all *a, b* types and *10x10 c* type problems) are referred as easy problems.

For hard problems, the proposed GA algorithm found LB values for 18 of the 24 problems while AIS found LB values for 17 of the 24 problems. Also for these hard problems GA found a better makespan value than AIS and B&B methods. For only two problems, GA could not reach the AIS's makespan value. These problems are represented in bold in Table 4.

The average % deviation from LB for GA is smaller than AIS and B&B methods'. There are 53 easy problems. Both of these methods, AIS and B&B, could not reach LB values for 6 of the problems. But the average % deviation from LB for GA algorithm is smaller than AIS and B&B methods'. In Table 5, the percentage of the solved problems and the average % deviation values for easy and hard problems are presented.

Table 5. Performances of three methods

	Easy problems		Hard Problems	
Method	% Solved	%Deviation	% Solved	% Deviation
GA	88.7	0.95	70.8	3.05
AIS	88.7	0.99	66.7	3.13
B & B	88.7	2.17	70.8	6.88

As it is clearly seen from Table 5, the least deviation belongs to GA. AIS is the second with a 0.08 % difference. B&B is the worst of all with almost two times larger deviation than the others.

Also the computational results are compared with the earlier study of Alaykıran et al. (2007). The average % deviations from LB due to the machine layout types are calculated for GA solutions and compared with the solution of Alaykıran et al.'s (2007) AS algorithm. The computational results are given in Table 6.

Table 6. The average % deviation from LB due to the machine layout types

Layout type		a	b	c	d
Average percentage deviations	AS	0.27	0.4	1.93	11.17
	GA	0.00	0.00	2.97	5.81

As it is seen in Table 6, the proposed GA found the optimal solutions for all *a* and *b* type problems, although Alaykıran et al.'s (2007) AS algorithm could not find optimal solutions. Also for *d* type problems the proposed GA found a smaller average % deviation from LB than Alaykıran et al.'s (2007) AS algorithm's. But for *c* type problems Alaykıran et al.'s (2007) AS algorithm found a smaller average % deviation from LB than the proposed GA.

The proposed GA can not be compared to the AIS and B&B according to CPU times because the configuration of the computers, in which the considered problems were solved, are different from one to another. The 1600 CPU time is used only a stopping parameters of GA.

4. Conclusion

In this paper, we propose an effective GA for HFS scheduling problems with the objective of minimizing makespan. The considered problem is a NP-Hard problem. Most of the studies to solve that problem are approximate methods rather than an exact method, which guarantees optimal solution. The test problems are benchmarking problems used in the literature. The percentage deviations from lower bounds are calculated. The findings are compared with another study that tested the same problems. We obtained better solutions with the proposed GA algorithm. When all problems are considered; the average deviation of the GA Algorithm is 1.50 % while the average deviations

of AIS and B&B are 1.657 % and 3.6 %, respectively. Also it can be seen in Table 4 that the CPU times of the GA are much smaller than AIS and B&B. The proposed GA is a good problem solving technique for a scheduling problem and may be used for some other industrial problems.

Acknowledgements

The authors would like to thank Jacques Carlier and Emmanuel Neron for the benchmark problems, solution files and any kind of help.

References

1. R. Linn, and W. Zhang, Hybrid Flow shop scheduling: A Survey, *Computers and Industrial Engineering.* **37** (1999) 57-61.
2. O. Engin, and A. Döyen, A New approach to solve hybrid flow shop scheduling problems by artificial immune system, *Future generation computer systems*. **20** (2004) 1083-1095.
3. J. N. D. Gupta, Two-stage hybrid flowshop scheduling problem, *Operational Research Society*, **39** (1988) 359-364.
4. M. A. Hong Wang, *A new model in designing neural network in optimization: A hybrid neural network approach to machine scheduling,* (Business Administration Graduate Program, The Ohio State University, Thesis, 1998).
5. T. S. Arthanari and K.G. Ramamurthy, An extension of two machines sequencing problem, *Opsearch*. **8** (1971) 10-22.
6. J. A. Hoogeveen, J. K. Lenstra, and B. Vettman, Minimizing the Makespan in a multiprocessor flow shop is strongly NP-Hard, *European Journal of Operational Research*. **89** (1996) 172-175.
7. L. Wang, L. Zhang, and D. Zhenga, An effective hybrid genetic algorithm for flow shop scheduling with limited buffers, Computers & Operations Research. **33** (2006) 2960–2971.
8. C. Oğuz, and M. F. Ercan, A Genetic Algorithm for Hybrid Flow-Shop Scheduling With Multiprocessor Tasks, *Journal of Scheduling*. **8** (2005) 323–351.
9. T. Li-Xin, W. Y. P. Source, and Z. Xuebao, Genetic descent algorithm for hybrid flow shop scheduling, *Automatica Sinica*. **28**(4) (2002) 637-641.
10. W. Xia, P. Hao, S. Zhang, and X. Xu, Hybrid flow shop scheduling using genetic algorithm, *Proceedings of the 3nd world congress on Intelligent control and Automation.* (2000) 537-541.
11. J. Carlier, and E. Neron, An exact method for solving the multiprocessor flowshop, *R.A.I.R.O-R.O.* **34** (2000) 1-25.
12. S. A. Brah, and J.L. Hunsucker, Branch and bound algorithm for flow shop with multiple processors, *European Journal of operations Research*. **51** (1991) 88-89.
13. D. L. Santos, J. L. Hunsucker, and D. E. Deal, Global lower bounds for flow shops with multiple processors, *European Journal of Operational Research*. **80** (1995) 112-120.
14. F. Riane, A. Artibs, and S.E. Elmaghraby, A Hybrid three stage flow shop problem: Efficient heuristics to minimize makespan, *European Journal of Operational Research* **109** (1998), 321-329
15. M. C. Portman, A. Vignier, D. Dardilhac, and D. Dezalay, Branch and Bound crossed with GA to solve hybrid flowshops, *European Journal of Operational Research*. **107** (1998) 389-400.
16. H. T. Jessen, and M. Weizhen, On-line algorithms for hybrid flow shop scheduling, *International conference on computer science and informatics*. (1998) 134-137.
17. O. Moursli and Y. Pochet, A branch and bound algorithm for the hybrid flow shop, *International journal of production economics.* **64** (2000) 113-125.
18. H. Soewandi, and S.E. Elmaghraby, Sequencing three stage flexible flowshops with identical machines to minimize makespan, *IIE Transactions*. **33** (2001) 985-983
19. E. Neron, P. Baptiste, and J. N. D. Gupta, Solving hybrid flow shop problem using energetic reasoning and global operations, *Omega The international journal of management science*, **29** (2001) 501-511.
20. L. Tang, W. Liu, and J. Liu, A Neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment, *Journal of Intelligent Manufacturing.* **16** (2005) 361-370.
21. L. Yang, H.G. Yu, and X.Y. Geng, Planning and scheduling algorithm based on TOC for complex hybrid flow shop problems, *Computer integrated manufacturing systems.* **11**(1) (2005) 97-103.
22. M. Zandieh, S. M. T. F. Ghami, and S. M. M., Husseini, An Immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times, *Applied Mathematics and Computation.* **180** (2006) 111-127
23. W. Zhong, X.F. Xu, and S. Deng, Evolutionary algorithm for solving multi-objective hybrid flow-shop scheduling problem, *Computer Integrated Manufacturing Systems.* **12**(8) (2006) 1227-1234.
24. H. Allaoui and A. Artiba Scheduling two stage hybrid flow shop with availability constraints, *Computers and Operations Research.* **33** (2006) 1399- 1419.
25. M. Haouari, L. Hidri, and A. Gharbi 2006, Optimal Scheduling of a two stage hybrid flow shop, *Math. Meth. Oper. Res.* **64** (2006) 107-124.
26. K. Alaykıran, O. Engin, and A. Döyen, Using ant colony optimization to solve hybrid flow shop scheduling problems, *Int. J. Adv. Manuf. Technol.* (2007) Article in press.
27. P. Caricato, A. Grieco, and D. Serino, TSP- based scheduling in a batch-wise hybrid flow shop, *Robotics and Computer-Integrated Manufacturing*. **23** (2007) 234-241.
28. A. Janiak, E. Kozan, M. Lichtenstein and C. Oğuz Metaheuristic approaches to hybrid flow shop scheduling problem with a cost related criterion,

International journal of production economics. **105** (2007) 407-424.
29. S. Vob, and A. Witt, Hybrid flow shop scheduling as a multi-mode multi project scheduling problem with batching requirements: A real world application, *Int. J. Production Economics.* **105** (2007) 445-458.
30. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, (Addison Wesley Publishing, The University of Alabama, 1989)
31. S. H. Yoon, J. A. Ventura, An application of genetic algorithms to lot-streaming flow shop scheduling, *IEE Transactions* **34** (2002) 77—787.
32. M. Gen, and R. Cheng, *Genetic Algorithms & Engineering Optimization*, (John Wiley & Sons, New York, 2000)
33. T. Yamada, and R. Nakano, A Genetic Algorithm Applicable To Large Scale Job Shop Problems, *Proceedings of The Second International Conference on Parallel Problem Solving from Nature*, (Elsevier Science Publishers, 1992).
34. İ. Kaya, O. Engin, A new approach to define sample size at attributes control chart in multistage process: An application in engine piston manufacturing process, *Journal of Materials Processing Technology*. **183** (2007) 38-48.