Application of Genetic Algorithms for a Tyre Production Scheduling Information System

Lin Liu
School of Management, Hefei University of Technology, Hefei 230009, China
liulinmail@163.com

Xinbao Liu, Hao Cheng, Ying Guo, Shanlin Yang
School of Management, Hefei University of Technology, Hefei 230009, China
lxinbao@mail.hf.ah.cn, chenghao2008@tom.com

***Abstract*—A optimization scheduling problem in a enterprise of manufacturing tyres is discussed in this paper. And this problem is reduced to a single machine scheduling problem to minimize setup times with batchs setup time depending on sequence. A method for solving tyre production scheduling problem using an effective adaptive hybrid genetic algorithm (AHGA) is proposed. We advance a novel operator (looping & cutting operator) to improve the mountain climbing ability of the genetic algorithm, and put forward adaptive probabilities of crossover and mutation based on information entropy. Computational results show that the proposed adaptive hybrid genetic algorithm is effective and robust.**

***Index Terms*—genetic algorithms, scheduling, single-machine, batch setup time**

I. Introduction

Most research on scheduling problems assumes that setup times are independent of the sequence of tasks on a machine. It is assumed that setup times are negligible or are added to the processing times of the tasks. However, significant setup times are incurred in some situations whenever a machine switches service from one task to another. In these cases, the machine processes many different jobs, and the setup time for a job depends on the job that has just finished processing before it. This kind of optimization scheduling problem was considered when the tyre production scheduling information system was designed in a enterprise. The enterprise manufactures a series of tyres. The process flow of producing tyres is that the steel ingots are rolled for shaping after they are cut and heated, shown in Figure1. The tyre production scheduling information system must make a optimal production plan for the jobs on production orders, in order to determine the optimization sequence that the jobs will be processed.

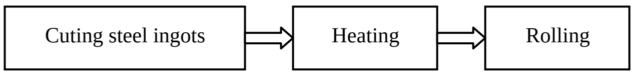

Figure 1. The process flow producing tyres

When the capability of cutting and heating stock is great enough, the processing sequence on rolling only need to be considered. Then this scheduling problem is regarded as a single-machine problem. In real-life, the manner of manufacturing tyres is batched job processing. The parts (jobs) in the same class are allocated into the same batch. The parts in the same batch are in one class. All the parts from the same batch must be processed consecutively, i.e., batchs cannot be split. The next batch of parts only begin to be processed after one batch finished. A setup time, which depends on both the previous and the current classes of jobs, is required for the changeover task, when the rolling mill is changed over from parts in one class to parts in a new class. The setup tasks include changing models and adjusting machines. The setup time lie on the type and number of the changed models. The moulds which are used in the parts in every class are different. So the setup time is different if the next batch is different, that is to say the setup times depend on sequence.

For example, there are four batchs (*b1*, *b2*, *b3*, *b4*) will be processed. The setup times that change one to another are shown as Table 1. The setup time is 2 hours if batch *b2* follows batch *b1*. It is just 1 hour if batch *b3* follows batch *b1*.

Supposing the processing times of all parts are constant, the scheduling problem to minimize maximum completion time of all jobs depends on the optimization of setup times between batchs completely. So the optimization objective of the scheduling problem is to minimize all setup times. Following the description of the three parameters, the scheduling problem is denoted by $1/r_j$, s-batch $/\sum r_j$. This is a typical problem of the optimization combination, which has been shown to be NP-hard. There are $n!$ schemes, if there are n batchs. The number of schemes will go up rapidly with n becoming large, viz. so-called "exponential burst". At present, it is unlikely for any algorithm to always find an optimal solution within a practical time limit. The better methods of solving the optimization scheduling problem are finding a near-optimal solution by heuristic, ant-colony algorithm, genetic algorithm, simulated annealing etc. D.Danneberg et al.[1] proposed different heuristic algorithms for flow shop scheduling problems with setup times and limited batch size. C. G. Skylab R. Gupta et al.[2] consider the problem of scheduling a single machine to minimize total tardiness with sequence dependent setup times. And they presented two algorithms, a problem space-based local search heuristic and a Greedy Randomized Adaptive Search Procedure (GRASP) for this problem. Ching-Jong Liao et al.[3] present an ant colony optimization (ACO) algorithm for a single machine scheduling problem with sequence-dependent setup times.

In this paper, we propose a kind of adaptive hybrid genetic algorithm (AHGA) to determine the optimal sequence that the jobs will be processed. The proposed AHGA has several features, including introducing a new operator to improve the mountain climbing ability of the genetic algorithm, and using information entropy for computation of sharing function, and employing information entropy for the adaptive probabilities formula of crossover and mutation. We have also applied the algorithm to the design of the tyre production scheduling information system. The outline of this paper is as follows. Section 2 starts with a brief description of the problem. Section 3 describes our proposed AHGA approach. Section 4 gives the results of computational experiments. In the end section 5 concludes with a summary of this research and a description of future work.

TABLE 1
SETUP TIMES (IN HOUR)

	To			
From	*b1*	*b2*	*b3*	*b4*
b1	0	2	1	1.5
b2	2	0	1.7	2.3
b3	1	1.7	0	2.5
b4	1.5	2.3	2.5	0

II. PROBLEM FORMULATION

Suppose

(1) The identical model of parts will be allocated into the same batch. The number of parts in each batch is determinate, before making production plan.

(2) The processing is continual in each batch, namely any other parts cannot begin to be processed before one batch is finished.

(3) The setup time of one batch is relevant to the model of parts before it.

(4) The setup time exhibit symmetry, if two different batchs counterchange, such as the setup time of batch *i* next to batch *j* equals that of batch *j* next to batch *i*.

Let $B = \{b_1, b_2, \ldots, b_n\}$ be the given set of product's batchs, $R_{(n\times n)} = \{r_{ij};\ r_{ij}\geqslant 0;\ i, j\in B\}$ be the matrix of setup times of batches, where r_{ij} is the setup time which is incurred when batch j immediately follows batch i. Let $S=\{s_1, s_2, \ldots, s_n\}$, a scheme, be a array of set B, and π denote the set of all arrays. The sum of setup times in scheme S is denoted by $f(S)$. Then this production scheduling problem is to solve the scheme, which satisfy

$$f(S_{opt}) = \min_{S\in\pi} f(S) = \min_{S\in\pi}\left(r_0 + \sum_{i=1}^{n-1} r_{s_i s_{i+1}}\right), \tag{1}$$

where r_0 is the first setup time before processing the first batch of jobs. In real-life production, the setup times required by various batchs are identity, if they are placed in the first position. So we can regard r_0 as a constant. The optimization objective given above is equal to

$$f'(S_{opt}) = \min_{S\in\pi} \sum_{i=1}^{n-1} r_{s_i s_{i+1}} \tag{2}$$

III. THE ALGORITHM

HollandJ[4] put forward a adaptive method to solve optimization problems by mimicking the evolving process of biological organisms in 1975. This method is genetic algorithm. Subsequently, people develop it continually. Now genetic algorithms have been applied to various optimization problems widely.

The standard genetic algorithm consists of the following steps[5].

Step 1. Determining the parameters of genetic algorithm: population size (represented as *POPSIZE*), crossover probability (represented as P_c), mutation probability (represented as P_m).

Step 2. Initialization: The initial population which size is *POPSIZE* is generated randomly, where every individual is a string type structure. And fitness valve of each individual is computed.

Step 3. Crossover: Two individuals selected based on the fitness function. Then their genes, which positions are selected randomly,are exchanged by P_c.

Step 4. Mutation: The individual is selected from the population, and it is recombined by using mechanisms of mutation.

Step 5. Reproduction: The fitness value of each chromosome in current generation is calculated. Then the regeneration times of each individual is computed, based on these values, some of them are selected for reproduction. The population size *POPSIZE* must be kept invariable after reproduction.

Step 6. If stopping criteria is met, then the individual with the maximum fitness value among current generation is regarded as the optimization result, else return step 3.

In this paper, we shall discuss the application of a kind of adaptive hybrid genetic algorithm (AHGA) to tyre production scheduling problem. The procedure of the new AHGA is described in Table 2. The implementation details are as follows.

A. Genetic code design

One of the important problem in using genetic algorithms is deciding on how chromosomes or solutions should be encoded. Consulted nine kinds of representations summarized in literature [6], we concluded to use the permutation representation based on batchs. For our problem, a chromosome represents a feasible solution of the problem. The size of a chromosome is n, each position in the chromosome corresponds to one batch number to schedule.

For example, if n=8, let the kth chromosome be p_k = [3, 2, 5, 8, 1, 6, 4, 7]. The chromosome means that the batchs are processed in the order of $b_3 \rightarrow b_2 \rightarrow b_5 \rightarrow b_8 \rightarrow b_1 \rightarrow b_6 \rightarrow b_4 \rightarrow b_7$.

TABLE 2
ADAPTIVE HYBRID GENETIC ALGORITHM (AHGA)

```
BEGIN
Generate the initial population $P(0)$, $|P(0)|=N$
FOR $i$=1 to $M$ DO
    FOR $j$=1 to N DO
        Select randomly two chromosomes in $P(i-1)$
        IF random<$P_c$ THEN
            Implement Crossover on both Parents
            Add the two offsprings to $P(i)$
        ENDIF
    ENDFOR
    FOR all chromosome $P_k \in P(i-1)$ DO
        IF random<$P_m$ THEN
            Implement Mutation of $P_k$ to obtain $P_k'$
            Add $P_k'$ to $P(i)$
        ENDIF
    ENDFOR
    FOR all chromosome $P_k' \in P(i-1)$ DO
        IF random<$P_m$ THEN
            Implement looping & cutting
            of $P_k'$ to obtain $P_k''$
            Replace $P_k'$ by $P_k''$ in $P(i)$
        ENDIF
    ENDFOR
    $P(i)=P(i) \cup P(i-1)$ (selection based on enlarged
    sampling space)
    Implement Selection in order to obtain population
    such that $|P(i)|=N$
ENDFOR
END
```

B. Initial population generation

Two questions must be considered about the initial population. One of them is size. It costs more time of computing during bigger size, while it is not sure that the optimization result is gotten. In this paper, let population size be N=2n. The other question is the way of generating individuals in the initial population. If every individual is generated randomly in the initial population, the capability of seeking the optimization result should be advanced, but the search process is hard. If the initial population is composed of individuals which are given by some heuristic algorithm, the average fitness value of the population should be increased. So convergence of performing the algorithm should be quick. But the search process is likely to fall into premature convergence to a local optima, because of a lack of the diversity of the individuals.

To give attention to convergence rate and premature convergence, a combination way is used for generating individuals in this paper. One better individual is made by a minimal neighborhood algorithm. Other individuals are generated randomly. The minimal neighborhood algorithm consists of the following steps.

step1. Look for jobs k and l in N. Let r_{kl}=min{r_{ij}; i, $j \in N$ }. K and l form the prime genes of individual P. A case that k follows l is same as a case that l follows k, because of $r_{k,l} = r_{l,k}$ here. We might as well suppose that l follows k.

step2. let $\alpha \leftarrow k, \beta \leftarrow l$.

step3. Delete jobs α and β from N.

step4. Look for job f in N. Let $r_{f\alpha} = \min\{r_{i\alpha}; i \in N\}$.
Look for job g in N. Let $r_{\beta g} = \min\{r_{\beta j}; j \in N\}$.

step5. If $r_{f\alpha} \leqslant r_{\beta g}$, put f into p, before α. Let $\alpha \leftarrow f$.
Delete jobs α from N.
Else, put g into p, follows β. Let $\beta \leftarrow g$.
Delete jobs β from N.

step6. If N is empty, obtain individual p.
Else, return step4.

C. Fitness and Selection

In GA, a fitness value is computed for each individual in the population, and the objective is to find a individual with the maximum fitness value. Inasmuch as the objective of this research is to minimize the total setup time, we take the fitness value of a chromosome of each chromosome to be the reciprocal of the objective function. namely the fitness value of a chromosome is determined by using the following equation:

$$Fitness = 1 \Big/ \sum_{i=1}^{n-1} r_{s_i s_{i+1}} \qquad (3)$$

A candidate solution with a small total setup time will lead to a chromosome with a large fitness value. As a result, the chromosome is given a greater chance to be selected as a parent chromosome to breed the offspring.

The selection may avoid loss of effective genes, and let high-powered individuals get greater probability of survival. Consequently, the constringency speed of the algorithm is expedited. In this paper, the Roulette Wheel Selection is chosen as the selection strategy, which is the method of direct proportion, and can select the individual in dircct proportion to its fitness value.

Let N be the population size, f_i be fitness value of individual i. Then the probability which individual i is selected is $P_i = f_i / \sum_{j=1}^{N} f_j$.

The niche method to suppress the similar individuals to maintain large diversity of the population is employed in selection operator. Fitness value of every individual is adjusted with sharing function, which is put forward by Goldberg et al.[7] in 1987. Selection operator is implemented according to the adjusted fitness value. The value of sharing function is large when genotype is similar, vice versa. The sharing degree of individual in population is defined by the information entropy theory in this paper. Suppose that there are N individuals in the population, and every individual is composed of n genes. Let R_j be a set of the values of the j th genes in N individuals. P_{ij} denote the percentage of the values of the j th gene lies in i th individual in R_j. Then the sharing degree of individual in population is defined as:

$$S_i = 1 \Big/ \sum_{j=1}^{n} P_{ij} \log_2 \frac{1}{P_{ij}}, \quad i=1, 2, \cdots, N \qquad (4)$$

The fitness value of every individual is recomputed as follows:

$$f_i^{'}(x) = f_i(x)\sum_{j=1}^{n} P_{ij}\log_2\frac{1}{P_{ij}},\ i=1, 2, \cdots, N \qquad (5)$$

D. The genetic operators

(1) Crossover operator

The evolution of the population is done through crossover and mutation. Through crossover, two parent members are exchanged partly and combined to form two new members. Crossover is the kernel operation in genetic algorithm. It is also a mechanism for chromosome diversification. Many crossover methods were brought forward by scholars.

Goldberg[8] proposed partially mapping crossover (PMX) in 1989. Davis[9] put forward order crossover (OX) and uniform order-based crossover in 1991. Reeves[10] proposed one-point order crossover (C1) in 1995. Croce[11] proposed linear order crossover (LOX) in 1995. Besides, other persons proposed position-based crossover (PX) resembling OX, cycle crossover (CX) etc. We discovered the OX is the most appropriate for this problem according to a lot of experimentation. The OX exercise is carried out in the following steps.

Step 1. Randomly choose two chromosomes, named P_1 and P_2.

Step 2. Select randomly two different gene positions i and j in P_1 and P_2. The close positions after i and j are just crossing positions, i.e. crossing areas are between gene positions i+1 and j.

Step 3. Copy the contents of the crossing areas to T_1 and T_2 respectively.

Step 4. Find out every gene positions x in P_1 according to mapping relation of the crossing areas, and let x be empty positions. The mapping relation is $p_x^1 = p_y^2$ (y=i+1, i+2, $\cdots$, j).

Analogously, find out every gene positions r in P_2 according to mapping relation of the crossing areas, and let r be empty positions. The mapping relation is $p_r^2 = p_y^1$ (y=i+1, i+2, $\cdots$, j).

Step 5. Move each gene toward left in P_1 or P_2 round until first empty gene reach the left side of the crossing area. Then move all empty genes into the crossing area, and move the genes that originally lie the crossing area to right simultaneously

Step 6. Interconvert the contents of T_1 and T_2, and put them into the crossing areas of P_1 and P_2 respectively. Generate new individuals $P_1^{'}$ and $P_2^{'}$.

For example, if choose P_1 = (4 9 2 8 1 7 5 3 6) and P_2= (5 7 6 3 2 4 9 1 8). Then the process employing OX to generate offspring is demonstrated in Figure 2.

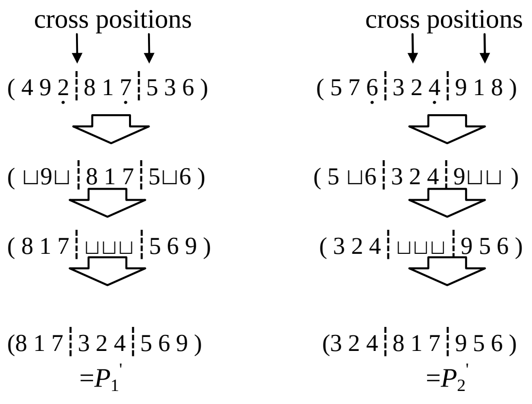

Figure 2. Order crossover process

(2) Mutation operator

The mutation operator is performed immediately after crossover. It is used to safeguard the search process from premature convergence to a local optima. The mutation operator rearrange the structure of a chromosome, and helps to increase the searching power. The probability of mutating a single gene is called the probability of mutation p_m.

The mutation operators used in genetic algorithm are, in the main, inversing mutation, swapping mutation and inserting mutation etc. We discovered the inserting mutation is the most appropriate for this problem according to a lot of experimentation.

The process of inserting mutation is shown as follows.

Step 1. Randomly choose two gene positions in individual.

Step 2. Insert one of two behind the other. Generate new individual $P_1^{''}$.

For example, if $P_1^{'}$ = (8 1 7 3 2 4 5 6 9). Then the process employing the inserting mutation to generate offspring is described in Figure 3.

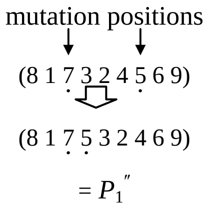

Figure 3. Inserting mutation process

(3) Probabilities of crossover and mutation

The given values of probabilities of crossover and mutation have an impact on genetic algorithm. Srinivas M and Patnaik L M[12] put forward adaptive probabilities of crossover and mutation. The probabilitie of crossover P_c and probabilitie of mutation P_m can automatically change by the fitness value of population. When fitness values of every individuals in the population go the same, P_c and P_m are able to increase automatically to escape from a local optima. While them are scattered, P_c and P_m are able to decrease automatically to favor the survival of good individuals. If fitness value of the individual is greater than average fitness value of the population, the less P_c and P_m are given to safeguard the good solution.

Contrarily, the greater P_c and P_m are given to expedite the generation of new individual. The formulae that were employed by Srinivas M and Patnaik L M[12] are demonstrated as follows.

$$P_c = \begin{cases} \dfrac{k_1(f_{\max} - f^{'})}{f_{\max} - f_{avg}} & f^{'} > f_{avg} \\ k_2 & f^{'} \le f_{avg} \end{cases} \quad (6)$$

$$P_m = \begin{cases} \dfrac{k_3(f_{\max} - f)}{f_{\max} - f_{avg}} & f > f_{avg} \\ k_4 & f \le f_{avg} \end{cases} \quad (7)$$

Where f_{max} is the most fitness value of individual in every population, f_{avg} is the average fitness value of the population, f' is the greater fitness value in two individuals chosen to implement crossover, f is the fitness value of individual chosen to implement mutation, k_1, k_2, k_3 and k_4 are the given constants.

It has been observed that the difference between the f_{max} and f_{avg} of the population likely to be less for a population that has converged to optimum solution than that for a population scattered in the solution space. Therefore, the rate of crossover and mutation operators should be varied depending on the value of the $f_{max} - f_{avg}$. For all the solutions of the population, which means that the solutions with high fitness values as well as the ones with low fitness values are subjected to the same level of crossover (or mutation) operation if P_c (or P_m) has the same value. This will certainly deteriorate the performance of GAs.[15] Later, Wu et al.[16] inserted an additional scheme into the original mutation scheme in order to prevent the deterioration of the performance as follows:

$$P_m = \begin{cases} \dfrac{\tilde{C}(f_{mut})k_3(f_{\max} - f_{mut})}{f_{\max} - f_{avg}} & f_{mut} > f_{avg} \\ k_4 & f_{mut} \le f_{avg} \end{cases} \quad (8)$$

where $\tilde{C}(f) = \dfrac{\sum_{j=1}^{off\_size} \left| f_{mut} - f_j \right|}{(off\_size - 1)\max_j\{| f_{mut} - f_j |\}}$

fmut is the best fitness value among the individuals to which the mutation with a rate P_m is applied. mut$\neq j$, f_j=all the individuals except f_{mut}.

The difference of the individuals in the population is calculated by the fitness values in the above formulations.

But this is not exact. And similitude between two individuals is not considered when the probabilities of crossover are calculated.

In this paper, we employ a sort of improved method to compute adaptive probabilities of crossover and mutation based on information entropy.

Suppose that there are N individuals in population, and each individual is composed of n genes. Let R_j be a set of the values of the j th genes in N individuals, b_{ij} be the times of the values of the j th gene lies in i th individual appear in R_j. P_{ij} denote the percentage of the values of the j th gene lies in i th individual in R_j. Then the information entropy of j th genes is defined as:

$$H_j = \sum_{i=1}^{N} \frac{P_{ij}}{b_{ij}} \log_2 \frac{1}{P_{ij}} \quad (9)$$

The population diversity is denoted as follows:

$$H = \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{N} \frac{P_{ij}}{b_{ij}} \log_2 \frac{1}{P_{ij}} \quad (10)$$

If the population has a trend of falling into local optimum, P_c and P_m will be increased in order to help the algorithm to escape from local optimal. Contrariwise, the population scatter in the solution space, P_c and P_m will be decreased. Furthermore, if fitness value of the individual is greater than average fitness value of the population, the less P_c and P_m are given to safeguard it into offspring. Whereas, the greater P_c and P_m are given to expedite the elimination of the solution.

If two individuals which are selected for mating are very similar, it is unnecessary that greater P_c is given, because the significance of crossover is little. Contrarily, if the difference of two individuals is great, the greater P_c must be given to help improve the search efficiency, by reason of a great probability of generating new individuals.

The original P_c and P_m are modified as follows based on the information entropy and above analyse.

$$P_c = \frac{P_{c2}H + P_{c1}(\log_2 N - H)}{\log_2 N} \quad (11)$$

$$P_m = \frac{P_{m2}H + P_{m1}(\log_2 N - H)}{\log_2 N} \quad (12)$$

where P_{c1} be the most value of probabilities of crossover, P_{m1} be the most value of probabilities of mutation, P_{c2} be the least value of probabilities of crossover, P_{m2} be the least value of probabilities of mutation. The curves of the adaptive probabilities of crossover and mutation is shown in Figure 4.

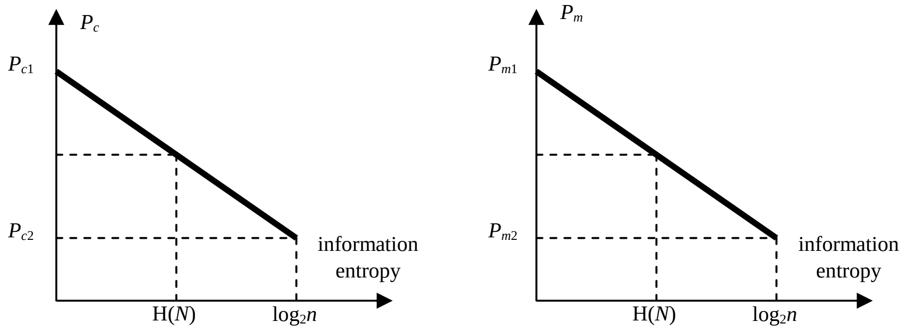

Figure 4 Adaptive probabilities of crossover and mutation

E. Looping & cutting operator

To advance searching efficiency, we devise a new operator (i.e. looping & cutting operator) and combine it with the genetic algorithm. The basic idea of this operator is to find the optimal order that the batchs are processed among all kinds of schemes, in which the relative processing orders are unchanging. The process of looping & cutting is shown as follows.

Step 1. Connect the head and end of chromosome into a loop.

Step 2. Find the position where the setup time is maximum in the loop.

Step 3. Cut the loop at the position where the setup time is maximum, then form a new chromosome.

Step 4. Replace the old chromosome with the new one.

For example, there is 4 batchs of jobs to be processed. Supposing that the setup times between batchs are shown in Table 3.

TABLE 3
THE SETUP TIMES BETWEEN BATCHS

	b_1	b_2	b_3	b_4
b_1	0	7	5	4
b_2	7	0	6	3
b_3	5	6	0	5
b_4	4	3	5	0

If choose T_{old} = (4 3 1 2). The chromosome means that the batchs are processed in the order of $b_4 \rightarrow b_3 \rightarrow b_1 \rightarrow b_2$. Connect the head and end of the chromosome into a loop:

$$\begin{tikzcd}4 \arrow[r] & 3 \arrow[r] & 1 \arrow[r] & 2 \arrow[lll, bend left]\end{tikzcd}$$

Then the setup time of $b_4 \rightarrow b_3$ is 5, the setup time of $b_3 \rightarrow b_1$ is 5, the setup time of $b_1 \rightarrow b_2$ is 7, the setup time of $b_2 \rightarrow b_4$ is 3. The position where the setup time is maximum in the loop is between b_1 and b_2. Cut the loop at where between b_1 and b_2. The new chromosome T_{new} = (2 4 3 1) is formed. By all appearances, the order of $b_2 \rightarrow b_4 \rightarrow b_3 \rightarrow b_1$ is the optimization in all schemes that relative orders are unaltered. This is shown in Table 4.

TABLE 4
ALL SCHEMES THAT RELATIVE ORDERS ARE UNALTERED

cutting point	schemes	total setup time	optimization
between b_2 and b_4	$b_4 \rightarrow b_3 \rightarrow b_1 \rightarrow b_2$	17	
between b_4 and b_3	$b_3 \rightarrow b_1 \rightarrow b_2 \rightarrow b_4$	15	
between b_3 and b_1	$b_1 \rightarrow b_2 \rightarrow b_4 \rightarrow b_3$	15	
between b_1 and b_2	$b_2 \rightarrow b_4 \rightarrow b_3 \rightarrow b_1$	13	*

F. Elitist preservation strategy[13]

The new individuals are generated continually by crossover and mutation in genetic algorithm. Although increasing good individuals appear in the population along with evolution, the crossover and mutation may destroy the best individual in current population because of their randomicity. This isn't what we expect, by reason of it can reduce the average fitness value of the population, and it is a bad influence on efficiency and convergence of performing the algorithm. We wish that the best individual found at each generation is stored unaltered in the next generation's population. To fill the purpose, we used an elitist preservation strategy, where the best individual in current population is stored and replace the worst individual in the population after crossover and mutation.

The process of the elitist preservation strategy operated is described below.

(1) Find out the best individual and the worst individual in current population.

(2) If fitness value of the best individual in the current population is greater than of the best individual so far, regard the best individual in the current population as the new best individual so far.

(3) Replace the worst individual with the best individual in the current population so far.

Elitist preservation strategy can be regarded as a part of the selection operation. The implementation of the strategy can promise the best individual gained so far will not be destroyed by the crossover and mutation operations. It is also an importance assurance that the genetic algorithm would converge.

3.7. Stopping criterion

The program is terminated when either the maximum number of generations is reached, or until the best individual of the population does not change for M consecutive generations. Where the maximum number of generations is 300, M=30+n/4, n is number of jobs .

IV. NUMERICAL COMPUTATION AND ANALYSIS

To compare the effectiveness of various crossover operators and mutation operators in the presented algorithm , we consider 6 instances accord with actual production.

The algorithm was implemented in Visual C++ 6.0 and the tests were run on a computer with a 2.8GHz Pentium 4 CPU on the MS Windows XP operating system.

A. The collaboration of diversified crossover operators with other operators

For reasons of comparison, we used 3 crossover operators respectively in the same algorithm, in which other operators are identic. Where let the population size N be double n. Employ the adaptive probabilities of crossover and mutation, the inserting mutation and the looping & cutting operator. The six cases were used in the simulation. Where the most size of the problem is 50 because the batchs of production processed are less than 50 in the enterprise every week. Every case is run 20 times randomly. The simulation results are shown as Table 5.

TABLE 5
THE EXPERIMENTAL RESULTS FOR 3 CROSSOVER OPERATORS

Prob. Size (n)	C^{**}	Crossover operator											
		PMX				CX				OX			
		C^*	t	%	t_e	C^*	t	%	t_e	C^*	t	%	t_e
5	4	4.2	4.89	5.00	47	4.9	5.07	22.50	50	4	5.60	0.00	5
10	9	9.7	5.50	7.78	85	13.5	5.37	50.00	73	9.6	5.27	6.67	44
20	19	25	6.43	31.58	137	28.6	7.01	50.53	166	19.2	8.46	1.05	76
30	29	38.7	11.97	33.45	272	43.9	12.22	51.38	208	30.3	15.44	4.48	161
40	39	57.6	12.89	47.82	322	60.6	15.21	55.38	247	40.3	16.98	3.33	141
50	49	74.5	17.85	52.04	296	72.7	21.00	48.37	324	50.8	24.33	3.67	137

n denotes the batch size, C^{**} is the best solution of the problem, C^* is the average best solution of 20 simulations, t is the average CPU time of 20 simulations, % is the relative deviations of C^* to C^{**}, t_e is the number of generations that C^* appears firstly.

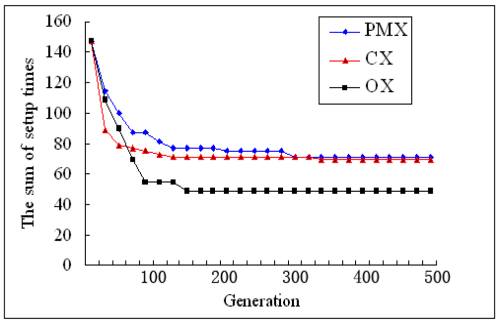

Figure 5. Convergence situations at various crossovers

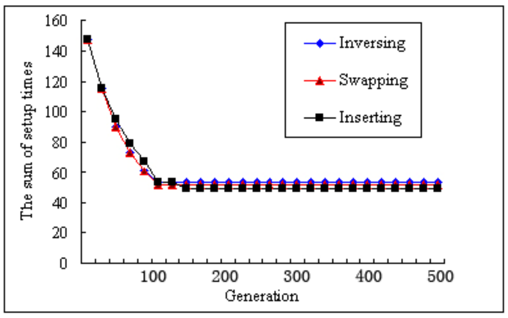

Figure 6. Convergence situations at various mutations

Figure 5 shows the convergence situations at various crossover operators when problem size is 50. It is easy to see that OX crossover operator is superior to PMX and CX especially in the large problem size if other conditions are same. So OX crossover operator will be selected in our algorithm.

Then we replaced the inserting mutation operator with the inversing mutation operator and the swapping mutation operator respectively. The simulation results resemble the above, they aren't shown here.

B. The collaboration of diversified mutation operators with other operators

For the same reason, to compare the effectiveness of various mutation operators, we also used 3 mutation operators respectively in the same algorithm. Where the crossover operator is OX, the adaptive probabilities of crossover and mutation, and employ the looping & cutting operator. other conditions are same as the above. The simulation results are shown as Table 6. Figure 6 shows that convergence situations at various mutation operators when problem size is 50.

The simulation results reveal that there is not large difference between the three mutation operators. But the inserting mutation operator is the best one. It always gives the best solution. So the inserting mutation operator is employed in our system.

C. The comparison between the AHGA and other GAs

In Table 7, we present a comparison between the adaptive hybrid genetic algorithm (AHGA) and the standard genetic algorithm (SGA) and adaptive genetic algorithms (AGA) with simulation model. AGA is a kind of adaptive genetic algorithm which was proposed by Srinivas M and Patnaik L M[12] in 1994. Figure 7 shows the convergence situations at three algorithms when problem size is 50.

TABLE 6
THE EXPERIMENTAL RESULTS FOR 3 MUTATION OPERATORS

Prob. Size (n)	C^{**}	Mutation operator											
		inversing				swapping				inserting			
		C^*	t	%	t_e	C^*	t	%	t_e	C^*	t	%	t_e
5	4	4	6.99	0.00	13	4	6.23	0.00	3	4	5.60	0.00	5
10	9	10.4	6.28	15.56	39	9.4	6.23	4.44	15	9.6	5.27	6.67	44
20	19	21.4	8.37	12.63	32	20.9	7.87	10.00	47	19.2	8.46	1.05	76
30	29	35.9	10.10	23.79	55	30.1	10.29	3.79	80	30	15.44	4.48	161
40	39	40	17.01	2.56	79	42.2	16.22	8.21	75	39.3	16.98	3.33	140
50	49	53.5	25.49	9.18	89	51.3	25.20	4.69	94	50.8	24.33	3.67	137

n denotes the batch size, C^{**} is the best solution of the problem, C^* is the average best solution of 20 simulations, t is the average CPU time of 20 simulations, % is the relative deviations of C^* to C^{**}, t_e is the number of generations that C^* appears firstly.

TABLE 7
THE EXPERIMENTAL RESULTS FOR 3 GAS

Prob. Size (n)	C^{**}	SGA				AGA				AHGA			
		C*	t	%	t_e	C*	t	%	t_e	C*	t	%	t_e
5	3.8	3.88	4.87	2.11	9	3.85	4.10	1.32	6	3.8	4.15	0.00	5
10	7.3	8.1	7.12	10.96	65	7.6	5.27	4.11	32	7.45	5.13	2.05	17
20	13.1	15.4	9.69	17.56	115	14	9.46	6.87	67	13.4	6.84	2.29	35
30	17.7	23.0	12.22	29.94	189	19.45	15.44	9.89	152	18.5	11.26	4.52	81
40	21.3	29.9	15.70	40.38	279	26.6	18.98	24.88	236	22.6	14.01	6.10	113
50	26.9	39.1	18.76	45.35	382	34.2	25.33	27.14	353	28.5	19.50	5.95	128

n denotes the batch size, C^{**} is the best solution of the problem, C^{*} is the average best solution of 20 simulations, t is the average CPU time of 20 simulations, % is the relative deviations of C^{*} to C^{**}, t_e is the number of generations that C^{*} appears firstly.

For all the problems, we obtain better results with the AHGA than with other GA tested at the best solution. The best solution appears earlier in the AHGA than other GA tested. Average CPU time of the AHGA is also shortest.

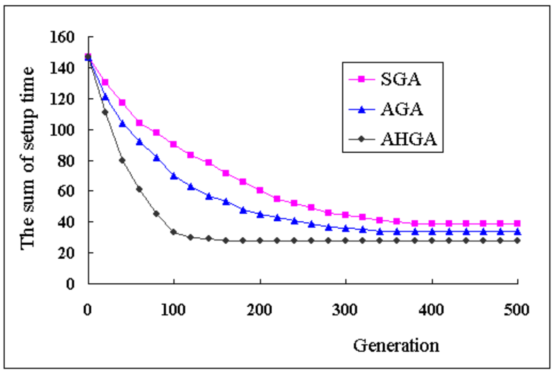

Figure 7. Convergence situations at 3GAs

V. CONCLUSION

The paper presents a adaptive hybrid genetic algorithm optimization approach for solving tyre production scheduling problem to minimize setup times with batchs setup time depending on sequence. The looping & cutting operator, we present, improves local search efficiency of the GA, improves optimization quality. It makes up for the shortage of SGA.

The concept of gene entropy is used for calculation of sharing function, adaptive probabilities of crossover and mutation, making the measures of the extent of individuals sharing and the diversity of the population more accurate, so as to improve the performance of the algorithm.

Now this AHGA has been employed to design MES by us in a enterprise which is the largest one of manufacturing tyres in the world. The result is satisfactory. The setup time is retrenched after the production scheduling is made with the MES.

We believe the idea of this AHGA, whih is proposed in this paper, could also be further applied to the permutation flow shop scheduling problem and the traveling salesman problem with success if it is improved ratherish.

ACKNOWLEDGEMENTS

Supported by the Decision Science and Technology Research Institute, Hefei University of Technology, Hefei, China is gratefully acknowledged. This research received financial assistance from the National High Technology Research and Development Program of China (viz. 863 Program) (project No. 2006AA04Z134).

The authors would like to put on record their appreciation to the anonymous referees for their valuable suggestions, which have enhanced the quality of the paper over its earlier versions.

Any errors in the paper are our own responsibility.

REFERENCES

[1] D.Danneberg et al. A Comparison of Heuristic Algorithms for Flow Shop Scheduling Problems with Setup Times and Limited Batch Size. Mathematical and Computer Modelling. 1999, vol. 29, pp. 101-126.

[2] Skylab R. Gupta, Jeffrey S. Smith. Algorithms for single machine total tardiness scheduling with sequence dependent setups. European Journal of Operational Research, 2006, vol. 175, pp. 722–739.

[3] Ching-Jong Liao, Hsiao-Chien Juan. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers & Operations Research, 2007, vol. 34, pp. 1899－1909.

[4] Holland J H. Adaption in natural and artificial systems. Cambridge, MA: MIT Press, 1975.

[5] Michalewicz Z. Genetic Algorihms+Data Structures=Evolution Programs. Beijing: Science Publishing House, 2000.

[6] Cheng Runwei, Gen Mitsuo, Tsujimura Yasuhiro. A tutorial survey of job-shop scheduling problems using genetic algorithms—PartⅡ: Hybrid genetic

search strategies. Computers & Industrial Engineering, 1999; vol. 36, pp. 343-364.

[7] Glodberg D E. Richardson J. Genetic Algorithms with Sharing for Multimodal Function Optimization. In: Proc. Of 2nd Int. Conf. on Genetic Algoriths, Lawrence Erlbaum Associates, 1987, pp. 41–49

[8] Goldberg D E. Genetic algorithms in search, optimization, and machine learning. MA: Addison-Wesley. 1989.

[9] Davis L. Hand book of genetic algorithms. New York:Van Nostrand Reinhold. 1991.

[10] Reeves CR. A genetic algorithms for flow shop sequencing. Computers and Operations Research. 1995, vol. 22, pp. 5-13.

[11] Croce F D, Tadei R, Volta G. A genetic algorithms for the job-shop problem. Computers and Operations Research. 1995, vol. 22, pp. 15-24.

[12] Srinivas M, Patnaik L M. Adaptive probabilities of crossover and mutation in genetical gorithm. IEEE Trans Systems Man and Cybernetics, 1994, vol. 24, pp. 656-667.

[13] WANG Wan-liang, WU Qi-di, SONG Yi. Modified Adaptive Genetic Algorithms for Solving Job-shop Scheduling Problems. Systems Engineering Theory & Practice, 2004, vol. 2, pp. 58-62.

[14] JosèFernando Gonçalves et al. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research, 2005, vol. 167, pp. 77-95.

[15] KwanWoo Kim et al. Hybrid genetic algorithm with adaptive abilities for resource-constrained multiple project scheduling. Computers in Industry, 2005, vol. 56, pp. 143–160.

[16] Q.H.Wu, Y.J. Cao, J.Y. Wen, Optimal reactive power dispatch using an adaptive genetic algorithm. Electrical Power & Energy Systems, 1998, vol. 20 (8), pp. 563–569.