Evolutionary Computation for Minimizing Makespan on Identical Machines with Mold Constraints

Tzung-Pei Hong[1, 2], Pei-Chen Sun[3], and Sheng-Shin Jou[3]
[1]Department of Computer Science and Information Engineering
National University of Kaohsiung
Kaohsiung, 811, Taiwan, R.O.C.
[2]Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, 804, Taiwan, R.O.C.
[3]Graduate Institute of Information and Computer Education
National Kaohsiung Normal University
Kaohsiung, 802, Taiwan, R.O.C.
e-mail: tphong@nuk.edu.tw, sun@nknucc.nknu.edu.tw, ssjou@nknucc.nknu.edu.tw

Abstract:- This paper focuses on minimizing makespan of identical-machines scheduling problems with mold constraints. In this kind of problems, jobs are non-preemptive with mold constraints and several identical machines are available. Feasible solutions are randomly generated and evolved by the proposed approach and an adjustment operator is designed to fill up the empty time slot due to the mold constraint. The proposed approach can also be easily extended to solve the same problem but with the fitness function of the total complete time. Experiments with and without the adjustment operator are made to analyze the performance of the proposed approach. Experimental results show that the adjustment operator does increase the performance of the scheduling.

Key-Words: - Identical Machine, Evolutionary Computation, Scheduling, Makespan, Adjustment Operator.

1 Introduction

Scheduling is an important process widely used in manufacturing, production, management, computer science, and among others. There are many kinds of scheduling problems in the literature. One of them is the identical-machine scheduling problem, in which scheduling jobs in identical machines [1][2][3][4][5][6][7] is considered. In this kind of problems, jobs are usually assumed to be non-preemptive with their own processing time and several identical machines are available to process the jobs. Scheduling jobs in identical machines for minimizing the makespan was proved as NP-hard [8].

The problem addressed in the paper is a special case of the above identical-machine scheduling problem. The problem to be solved assumes each job needs to use one particular mold to perform its work in addition to the above constraints. Different jobs may need different or the same molds. There is only one mold for each kind. The purpose is to minimize the makespan of the whole schedule.

The above problem is hard to solve, such that evolutionary computation is used here in order to find a feasible solution. Besides, a scheduling may need to be adjusted to fill up the empty time slot due to mold constraint. Experiments were also made to analyze the performance of the proposed algorithm.

The remainder of this paper is organized as follows. Related scheduling algorithms are briefly reviewed in Section 2. The proposed scheduling algorithm for identical machines with mold constraints is proposed Section 3. The experiments conducted to make the comparative analysis for the proposed algorithm are presented in Section 4. An extension to total completion time is stated in Section 5. Finally, conclusion is given in Section 6.

2 Related Works

In the past, many researches have studied the identical parallel-machine scheduling. Park and Kim [1] presented a local search heuristic, which combined simulated annealing and tabu search, to solve this problem with ready times and due dates. Lee and Pinedo combined a heuristic and the simulated annealing method to schedule jobs with sequence-dependent setup times of machines [2]. Armentano and Yamashita [4] introduced the tabu search approach for scheduling to minimize mean tardiness. Jeong and Kim [7] considered

earliness-tardiness penalties and space limits and proposed a heuristic algorithm.

As to using genetic algorithms to solve this kind of problems, Funda and Gunduz considered scheduling with earliness and tardiness penalties [3]. Cheng et.al used genetic algorithms to solve the scheduling in identical machines with minimum tardiness [5]. Liu and Wu [6] studied a genetic algorithm to find nearly minimal makespan on identical parallel machines. Low and Yeh [9] proposed a genetic-algorithm-based heuristic for scheduling problems with setup, processing and removal times. Allahverdi [10] surveyed papers on scheduling problems with setup times or costs.

Several studies discussed scheduling with mold constraints. For example, Chen and Wang proposed a heuristic algorithm to solve scheduling with mold constraints [11]. Chen and Wu then proposed their algorithm based on threshold-accepting methods, tabu lists, and scheduling-improvement procedure [12]. Hong et al. considered some related constraints (e.g. processing time, due date and mold constraint) on unrelated parallel machines and proposed some heuristic algorithms [13].

As mentioned above, the scheduling problem on identical machines is NP-hard. No algorithms can find the optimal solution in polynomial time. In this paper, we propose a GA-based approach to find a nearly optimal makespan of the scheduling problem with mold constraints on identical machines.

3 The Proposed Scheduling Algorithm

In this section, a GA-based algorithm is proposed to solve the above scheduling problem in which jobs are non-preemptive and several identical machines are available. Besides, there are a set of different molds and each job needs to use one mold to perform its work. Different jobs may need the same or different molds. Each job has its processing time. When a mold is installed on a machine, some set up time is needed. The purpose is to make the makespan minimum.

Assume there are ten jobs (job 1 to job 10) and two machines (machine *A* and *B*) for scheduling. Table 1 shows the processing time of each job. There are four different molds and each job needs a mold. The problem is to find a schedule with the makespan minimum. The proposed approach is based on the genetic algorithms with several heuristic procedures designed as well to find appropriate solutions. The assumptions and the notation used in the proposed approach are first described below.

3.1 Assumptions and Notation

The assumptions made in this paper are stated as follows.

- Each job has its own processing time and suitable mold.
- Each job can be executed on one of the identical parallel machines with one mold.
- Each mold can only be loaded on one machine at a time.
- Each mold needs some set-up time to be equipped on a machine.
- Each job is non-preemptive.

The notation used in the paper is listed as follows.

n: the number of jobs;
k: the number of identical machines;
h: the number of molds;
j_i: the *i-th* job;
m_i: the *i-th* machine;
c_j: the complete time of the *j-th* job;
S: the set-up time

3.2 Chromosome Representation

It is important to encode a possible solution into a chromosome representation for genetic algorithms to work. In this paper, a feasible schedule is a possible solution. Several encoding approaches were proposed in the past [3][5][6]. The encoding scheme adopted in this paper is based on Funda and Gunduz's work [3]. That is, each gene includes both a job code and a machine code, representing the job is performed on the machine. For a problem with *n* jobs, a chromosome is thus composed of *n* genes. The encoding scheme can be represented by Figure 1, in which the *i*-th gene, j_{ui}-m_{vi}, represents that job j_{ui} is executed on machine m_{vi}.

For any two different values of *i* and *j*, j_{ui} and j_{uj} must be different since the scheduling problem considered here is non-preemptive. m_{vi} and m_{vj} may, however, be the same since several jobs can be performed on a machine at different time. An example is given below to demonstrate the encoding scheme of a schedule.

Assume there are ten jobs (jobs 1 to 10) and two machines (machines *A* and *B*) for scheduling. The chromosome shown in Figure 2 represents job 3 is performed on machine *B*, job 4 is performed on machine *A*, and so on.

When the representation is used to represent a schedule, the jobs performed on the same machine are aggregated together according to their original order in the chromosome. For the above example, the scheduling results are shown in Figure 3.

Table 1: The processing time and the required mold of each job

Job	1	2	3	4	5	6	7	8	9	10
Processing time	6	12	9	19	15	12	11	13	16	12
Mold	1	3	2	2	4	3	3	1	4	2

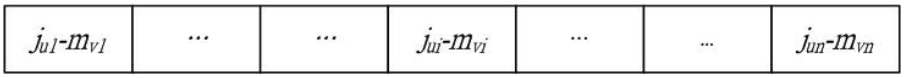

Fig.1: The encoding scheme

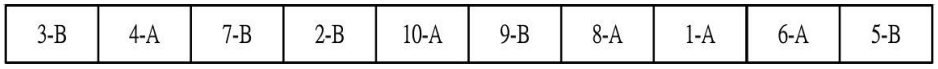

Fig.2: An example for the representation of a chromosome

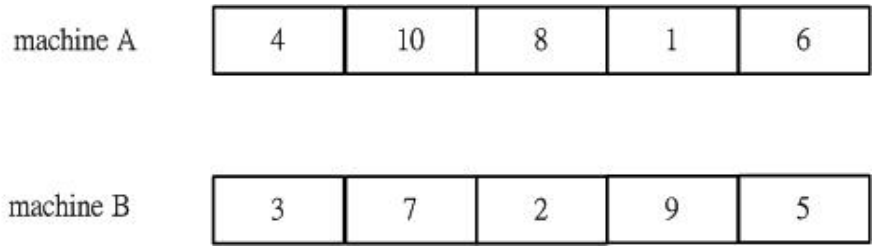

Fig.3: The jobs aggregated together on the same machine

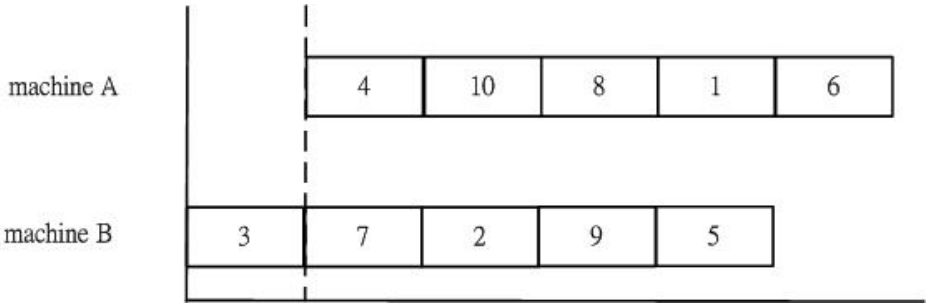

Fig.4: Job 4 is postponed until Job 3 is finished

Besides, in our problem, each job needs a mold to finish its task. Different jobs on different machines may use the same mold. Because this paper assumes there is only one piece for each type of mold, the latter job appearing in the original order of the chromosome will be postponed until the former job is finished if they use the same mold on different machines at the same time period. Take the schedule in Figure 3 as an example. If job 3 on machine *B* and job 4 on machine *A* use the same mold, then job 4 will be postponed until job 3 is finished. The results are shown in Figure 4.

In this case, some jobs on machine *A* may be moved forward to the empty time slot from the postponement for improving the scheduling performance. This can be done by the proposed adjustment operator, which will be described later. The adopted encoding scheme is then simple, but can represent the complicated scheduling results.

According to the proposed representation, each chromosome consists of a sequence of job-machine pairs. This representation then allows genetic operators (defined later) to search for appropriate scheduling solutions.

3.3 Initial Population

A genetic algorithm requires a population of feasible solutions to be initialized and then updated during the evolution process. According to the above

encoding scheme, each chromosome can be mapped to its only corresponding feasible schedule as long as no two jobs in the chromosome have the same job ID. That is, any job can only appear once in the chromosome. Each individual in the initial population can then be randomly generated with the above constraint.

3.4 Initial Population

The jobs in the machine with the empty time slot are then checked in sequence. A job is moved forward if the mold constraint is not violated and its processing time plus setup time (if needed) is less than the time interval of the slot. After the job is moved, the original time slot may still have space and the moved job also causes an empty time slot. The other jobs are then processed repeated to reduce the empty time intervals in the slots. Take the schedule in Figure 5 as an example. It is nearly the same as Figure 3 except with length representing the set-up (S) time and the processing time of the jobs.

From Figure 5, it can be seen that job 4 and job 3 use the same mold. Job 4 is then postponed with a empty time slot generated as shown in Figure 6. Job 10 also needs to use mold 2 and thus cannot be moved forward. Job 8 cannot be moved forward since its processing time is larger than the time interval of the empty slot. The sum of the processing time and the set-up time of job 1 is less than the interval. It is then moved forward.

After job 1 has been moved, the original time interval for job 1 becomes empty in Figure 7. Job 6 is then moved forward to the time slot. The final results are shown in Figure 8.

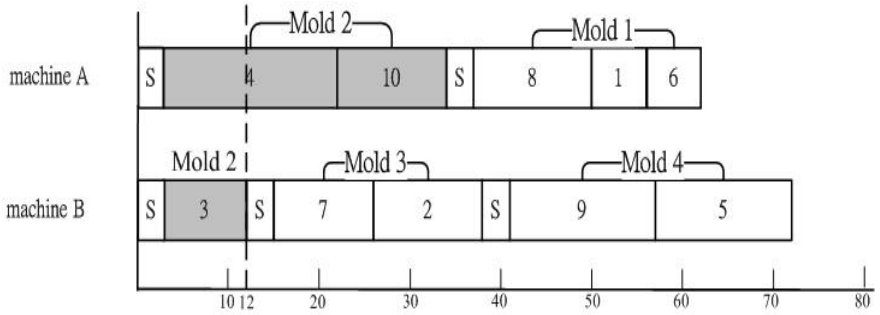

Fig.5: Jobs 3 and 4 use the same mold in the same time period

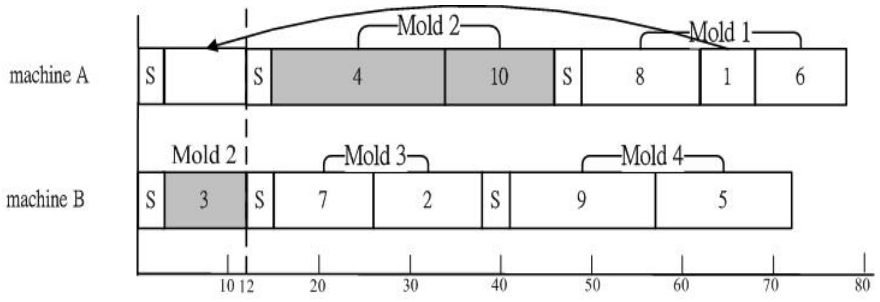

Fig.6: Job 1 is moved forward

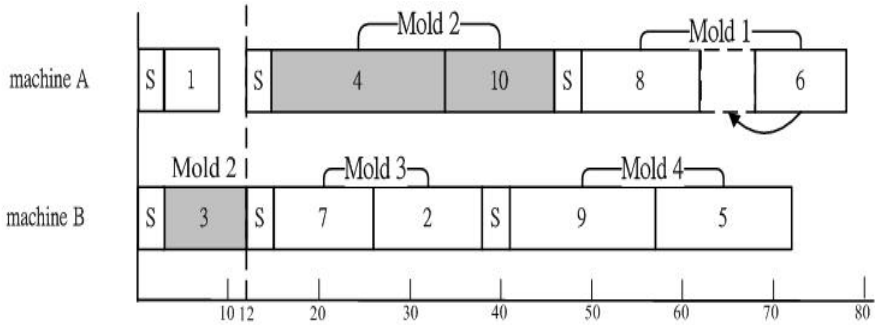

Fig.7: Job 6 is moved forward to the time slot

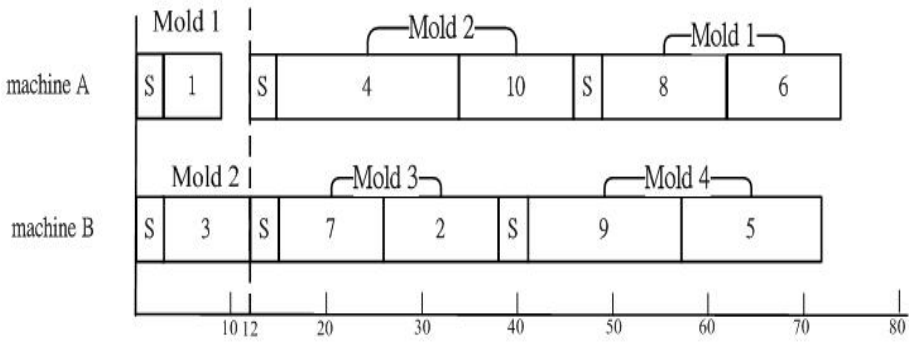

Fig.8: The final results after job 6 is moved forward

3.5 Fitness and Selection

A fitness function must be designed and used to evaluate the performance (fitness) of each individual chromosome in the population. Besides, in order to develop a good schedule from an initial population, the genetic algorithm selects *parent* schedules with high fitness values for mating. In this paper, since the purpose is to minimum the makespan of the whole schedule, the following formula is adopted as the fitness function:

$$f = \max(c_1, c_2, ..., c_n),$$

where c_j is the completion time of job j and n is the number of jobs.

For example, assume there are ten jobs to be scheduled on two machines (A and B) and each job has its processing time and processing time as shown in Table 1. Also assume the set-up time for all the molds are the same and equal to 3. For the schedule in Figure 9, its fitness value is calculated as follows:

$$f = \max(c_1, c_2, .., c_n),$$
$$= \max(6, 38, 12, 34, 72, 74, 26, 62, 57, 46)$$
$$= 74$$

3.6 The Crossover Operators

Crossover and mutation operators are very important to the success of specific GA applications. In this paper, the *combined order and position-based crossover* (OPX) is used [14].

The OPX crossover operator randomly selects some gene positions from the first parent and then copies the jobs to the same positions of the offspring. It then removes the selected jobs from the second parent and then copies in sequence the remaining jobs in the second parent to the offspring. The same action is repeated for the two parents with their roles interchanged. An example is given below to demonstrate the OPX crossover operation.

Assume the two parent chromosomes C_1 and C_2 are selected for crossover. Assume the first, the fourth, the seventh and the ninth positions are selected. The jobs in these positions of C_1 are then copied into the offspring in Figure 10. Then, the remaining jobs in C_2 are then copied in sequence into the offspring in Figure 11. After that, C_1 and C_2 are interchanged and another offspring is generated. Thus two selected parents will generate two offspring. The results are shown in Figure 12.

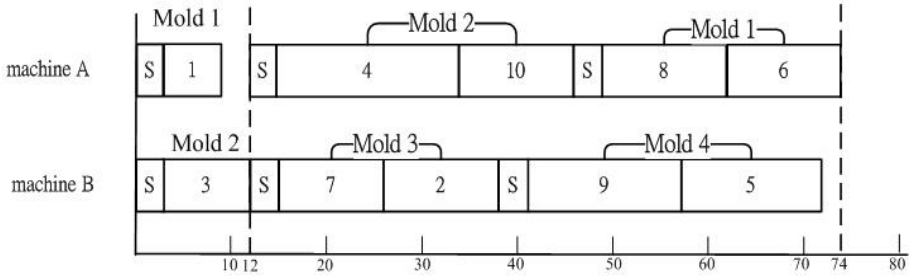

Fig.9: The final scheduling result in the example

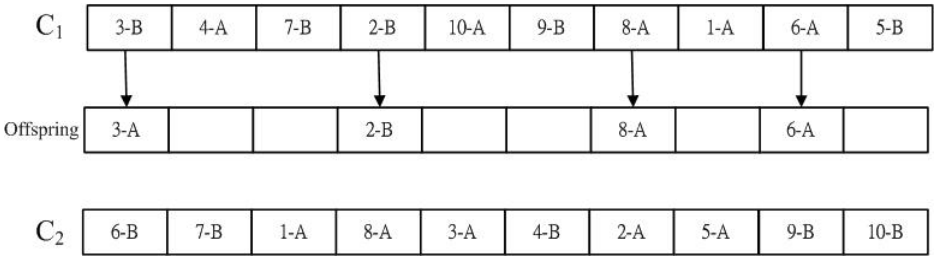

Fig.10: The OPX crossover operation according to the first parent

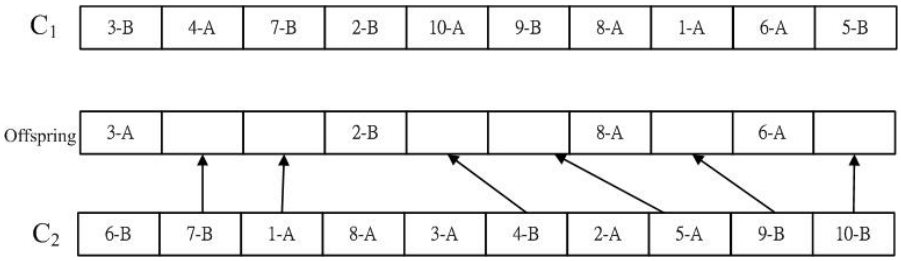

Fig.11: The offspring generated from the OPX crossover operation

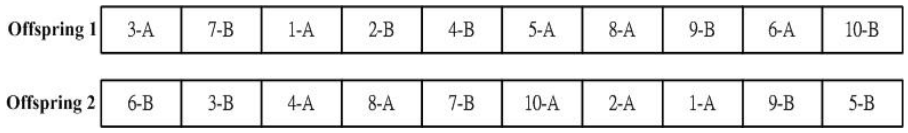

Fig.12: Two selected parents generate two offspring

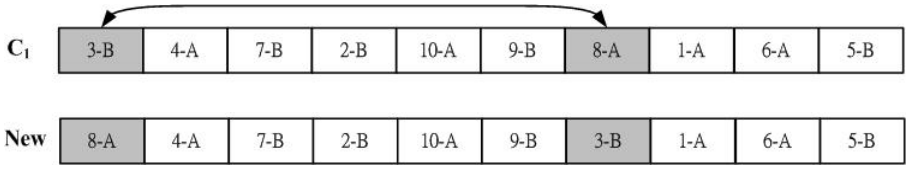

Fig.13: An example of the swap mutation operation

3.7 The Mutation Operator

In this paper, the swap mutation [14] is used for increasing the diversity of the chromosome. The swap mutation is first introduced below. It randomly selects two genes from a chromosome and then swaps them. If the two genes are processed on the same machine, their execution orders are then changed. If the two genes are processed on different machines, then both their orders and machines are changed. A new schedule is thus generated by the operator. An example is given below to demonstrate the swap mutation operation.

Assume the chromosome C_1 of Figure 13 is selected for mutation. The first and the seventh positions are selected for swapping. Job 3 executed on machine *B* is then exchanged with job 8 on machine *A*. The results are shown at the bottom of Figure 13.

4 Experiments

This section reports on experiments made to show the performance of the proposed GA-based algorithm for scheduling on identical machines with the mold constraints. They were respectively implemented by Dev C++ on an AMD Athlon XP 3000+ PC. Three parameters were considered to define the problem. They were number of jobs, number of identical machines, and number of molds. In the experiments, the job numbers were set at 20, 30, 40 and 50, the machine number was set at 9, and the mold number was set at 9. Each job is then randomly assigned a mold and its processing time.

Some related parameters for the proposed GA-based scheduling algorithm are shown as follows. The population size was set at 50, the number of generations at 250, the crossover probability *pc* at 0.7, and the mutation probability *pm* at 0.05.

The results of the proposed GA-based scheduling algorithms with and without the adjustment operator were compared. The makespans of the two GA-based scheduling algorithms for different numbers of jobs are shown in Figure 14.

From Figure 14, it is easily seen that the proposed algorithm with the adjustment operator got a better result than that without the adjustment operator. Next, the makespans of the two GA-based scheduling algorithms for different numbers of machines are shown in Figure 15. From Figure 15, it can also be easily seen that the proposed algorithm with the adjustment operator got a better result than that without the adjustment operator for any number of machines. Next, the makespans of the two GA-based scheduling algorithms for different numbers of molds are shown in Figure 16. From Figures 14 to 16, we find the adjustment operator can increase the performance of the proposed approach. The adjustment operator did reduce the processing time needed on the machines with the mold constraint to achieve the nearly minimum makespan.

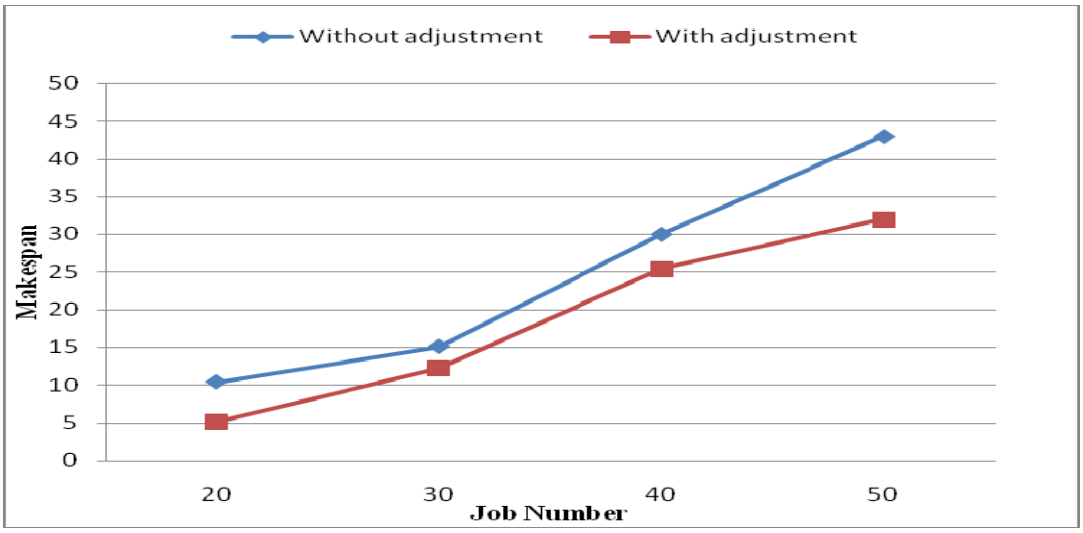

Fig.14: The makespans of the schedules by the two GA-based scheduling algorithms for different numbers of jobs

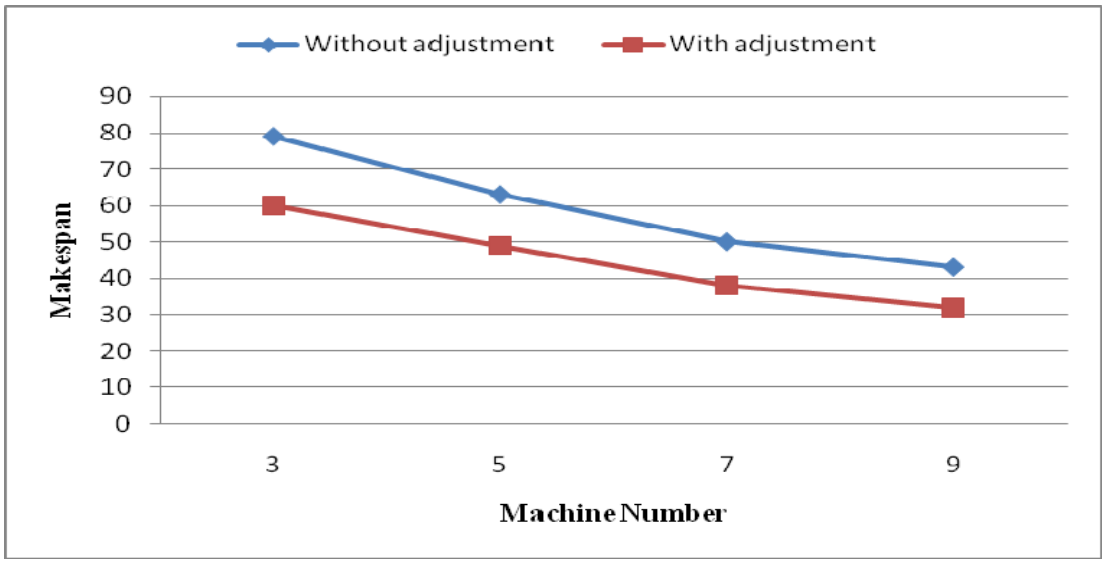

Fig.15: The makespans of the schedules by the two GA-based scheduling algorithms for different numbers of machines

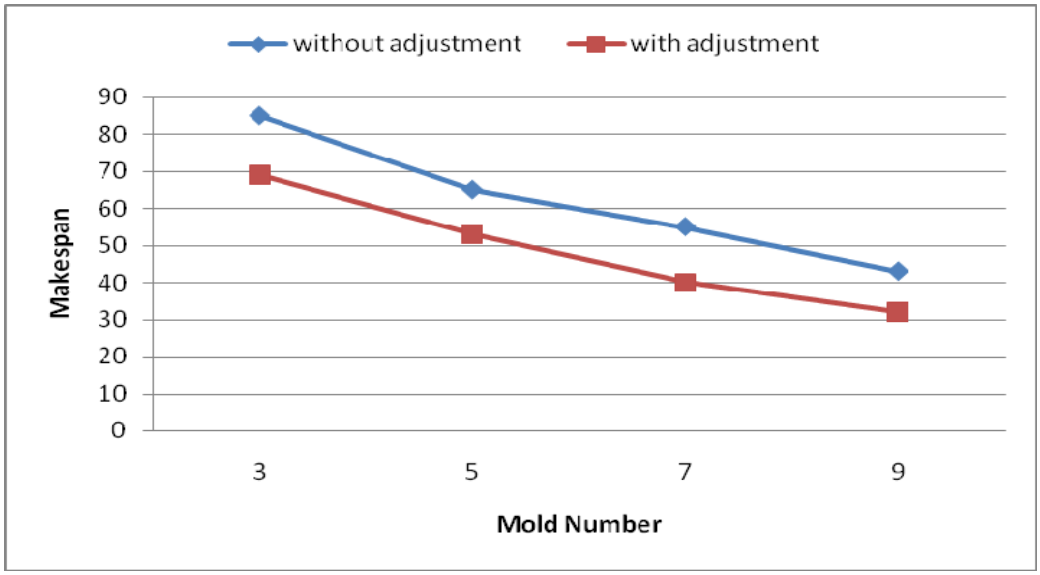

Fig.16. The makespans of the schedules by the two GA-based scheduling algorithms for different numbers of molds

5 Extension

The above scheme can be easily extended to solve the scheduling problem with minimizing the total completion time. The fitness function is then modified as follows.

$$f = \sum_{j=1}^{n} c_j,$$

where c_j is the completion time of job j and n is the number of jobs.

Again, take the data in Table 1 and the schedule in Figure 9 as an example. Its fitness value for the total completion time is calculated as follows:

$$f = \sum_{j=1}^{n} c_j$$
$$= 6 + 38 + 12 + \ldots + 46$$
$$= 427$$

The experimental environments for minimizing the total completion time are similar to the above. The total completion time of the two GA-based scheduling algorithms for different numbers of jobs is shown in Figure 17.

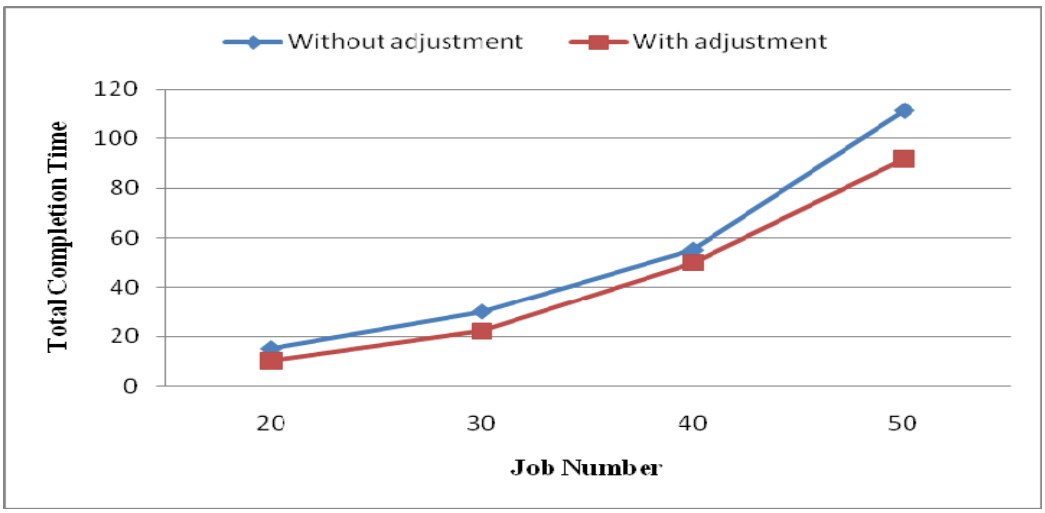

Fig.17: The total completion time of the schedules by the two GA-based scheduling algorithms for different numbers of jobs

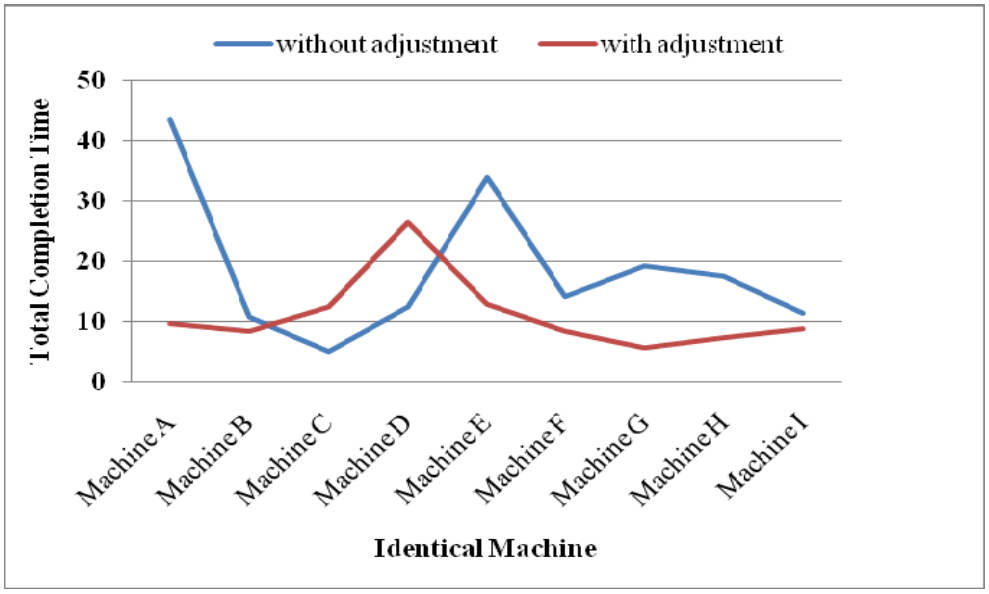

Fig.18: The total completion time on each machine by the two GA-based scheduling algorithms for 50 jobs

The total completion time on each machine by the two GA-based scheduling algorithms for 50 jobs is shown in Fig. 18. It can be seen from the figure that the results by the adjustment operator will generate a more balanced load distribution among the machines than those without the adjustment operator. The adjustment operator thus plays an important role on the proposed approach.

6 Conclusion

In this paper, we have proposed a GA-based approach to solve the scheduling problem with the minimum makespan on identical machines with mold constraints. The adjustment operator has also been adopted in the proposed approach to increase the performance due to the mold constraints. Experimental results also show the effectiveness of the adjustment operator. The proposed approach can also be easily extended to solve the same problem but with the fitness function of the total complete time. Similar experiments results are obtained.

In the future, we will consider attempt to solve the more scheduling problems on identical machine with mold constraint by using different search strategies.

References:

[1] M. W. Park and Y. D. Kim, Search heuristics for a parallel machine scheduling problem with ready times and due dates, *Computers & Industrial Engineering*, vol. 33, 1997, pp. 793-796.

[2] Y. H. Lee and M. Pinedo, Scheduling Jobs on Parallel Machines with Sequence-Dependent Family Set-up Times, *European Journal of Operational Research*, vol. 100, 1997, pp. 464-474.

[3] S. S. Funda and U. Gunduz, Parallel machine scheduling with earliness and tardiness penalties, *Computer & Operation Research*, vol. 26, 1999, pp. 773-787.

[4] V. A. Armentano and D. S. Yamashita, Tabu search for scheduling on identical parallel machines to minimize mean tardiness, *The Journal of Intelligent Manufacturing*, vol. 11, 2000, pp. 453-460.

[5] R. Cheng, M. Gen, and T. Tozawa, Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms, *The 17th International Conference on Computers and Industrial Engineering*, 1995, pp. 513-517

[6] M. Liu and C. Wu, A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines, *Artificial Intelligence in Engineering*, vol. 13, 1999, pp. 399-403.

[7] S. J. Jeong and K. S. Kim, Parallel machine scheduling with earliness-tardiness penalties and space limits, *The International Journal of*

Advanced Manufacturing Technology, vol. 37, 2008, pp. 793-802.

[8] P. Brucker, *Scheduling Algorithms*, Springer, 1995.

[9] C. Low and Y. Yeh, Genetic algorithm-based heuristics for an open shop scheduling problem with setup, processing, and removal times separated, *Robotics and Computer-Integrated Manufacturing*, vol. 25, 2009, pp. 314-322.

[10] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, A survey of scheduling problems with setup times or costs, *European Journal of Operational Research*, vol. 187, 2008, pp. 985-1032.

[11] X. Cheng and C. Wang, Scheduling on the parallel machines with mould constraint, *The 38th SICE Annual*, 1999, pp. 1167-1170.

[12] J. F. Chen and T. H. Wu, Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints, *The International Journal of Management Science*, vol. 34, 2006 , pp. 81-89.

[13] T. P. Hong, P. C. Sun, and S. D. Li, A heuristic algorithm for the scheduling problem of parallel machines with mold constraints, *The 7th WSEAS International Conference on Applied Computer & Applied Computational Science*, 2008.

[14] J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, and F. Werner, comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria, *Computer & Operation Research*, vol. 36, 2009, pp. 358-378.

[15] S. M. Alaoui, O. Frieder, and T. El-Ghazawi, A parallel genetic algorithm for task mapping on parallel machines, *Lecture Notes in Computer Science*, vol. 1586, 1999, pp. 201-209.