Improvements to Genetic Algorithm for Flexible Job Shop Scheduling with Overlapping in Operations

Yiyong He, Wei Weng, Shigeru Fujimura
Graduate School of Information, Production and Systems, Waseda University
2-7 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan

***Abstract*—Flexible job-shop scheduling problem (FJSP) is an extended job-shop scheduling problem. FJSP allows an operation to be processed by several different machines. FJSP with overlapping in operations means that each operation is divided into several sublots. Sublots are processed and transferred separately without waiting for the entire operation to be processed. In previous research, a mathematical model was developed and a genetic algorithm proposed to solve this problem. In this study, we try to improve the procedure of previous research to achieve better results. The proposed improvements were tested on some benchmark problems and compared with the results obtained by previous research.**

Keywords—flexible job shop, scheduling, overlapping in operation, genetic algorithm

I. Introduction

The flexible job-shop scheduling problem (FJSP) is an extension of the classical job-shop scheduling problem. It allows an operation to be performed by several different machines rather than only one machine, which makes the job-shop scheduling problem closer to actual production situation [1]. FJSP with overlapping in operations means that an operation is divided into multiple sublots and each sublot is treated separately and transferred to the next processing stage once it is completed. Production with sublots is a new assumption in FJSP, which is proposed by Fantuhan et al. [2], and further discussed by Demir and Isleyen [3]. To consider this assumption, an approach named overlapping in operations is used. In this approach, in order to start a successor operation, it is not necessary to finish its predecessor operation completely [4].

FJSP with overlapping in operations is found in multi-stage manufacturing systems, especially in the semiconductor manufacturing industry. Factories of semiconductor manufacturing industries usually consist of multi-purpose machines. Hence the efficiency of scheduling has an impact on the performance of total production process.

In the previous research of Demir and Isleyen [3], an effective genetic algorithm (GA) was proposed for solving large sized FJSP with overlapping in operations. GA is also applied in research by Chen et al. [5] and research by Pezzella et al. [6]. Job-shop scheduling problem is a well-known NP-hard problem [7]. Since FJSP with overlapping in operations is dealing with more complex condition than the job-shop scheduling problem, FJSP with overlapping in operations is NP-hard, too. The reason why we choose GA as our main method to solve this problem is that GA can get a good enough result in a reasonable time. Moreover, in recent years, it is proved that meta-heuristics such as simulated annealing, tabu search and GA can led to better results than classical dispatching rules or greedy heuristics [8][9][10].

In this research, some improvements based on the previous research [3] are proposed. The improvements are mainly intended to increase diversity and avoid falling into local optimum. We improve the selection operator of GA by not allowing the same individual to be selected multiple times. We modify the crossover operator by maintaining the feasibility relationship between machine selection and sequencing of operations. We improve the mutation operator by reassigning operations that are in the critical path so as to increase the chance of changing the makespan of a schedule. The proposed improvements are tested on a set of benchmark problems and the results are compared with results of the previous research.

The remainder of this paper is organized as follows: Section 2 gives the problem statement. Section 3 introduces the original GA in the previous research and the proposed improved GA. Section 4 presents the computational simulations and compares the obtained results with those of the previous research. Finally, in Section 5, some conclusions and direction for future work are given.

II. Problem Statement

The target problem of this paper is the same as that in a previous research [3]. The problem is to organize the processing of n jobs on m machines with the objective of minimizing the makespan, i.e., total completion time of all jobs. Let M denote the m machines $M = \{M_1, \ldots, M_m\}$ and J denote the n jobs, $J = \{J_1. \ldots. J_n\}$. Each job Ji consists of n_i operations in a specific sequence: $O_{i1}, O_{i2}, \ldots, O_{ini}$. Operation O_{ij} means operation j of job i. Ji is completed when all of its operations are finished one after another in the predefined order. Each operation O_{ij} can be processed on machine k with processing time t_{kij}, as long as machine k belongs to a subset

Wei Weng is the corresponding author (email: wengwei@toki.waseda.jp).

$M_{ij} \subseteq M$, which is the set of available machines for operation O_{ij}. No idle time between two adjacent sublots of the same operation is permitted and all sublots that belong to the same operation must be operated on the same machine one after another. Fig.1 shows 3 different cases of overlapping in operations, of which the first case is not allowed in this study. In the first case, every sublot of O_{ij+1} starts as long as a sublot in O_{ij} is finished, which leads to some idle time between sublots of O_{ij+1}. In the second and third cases, this problem is avoided.

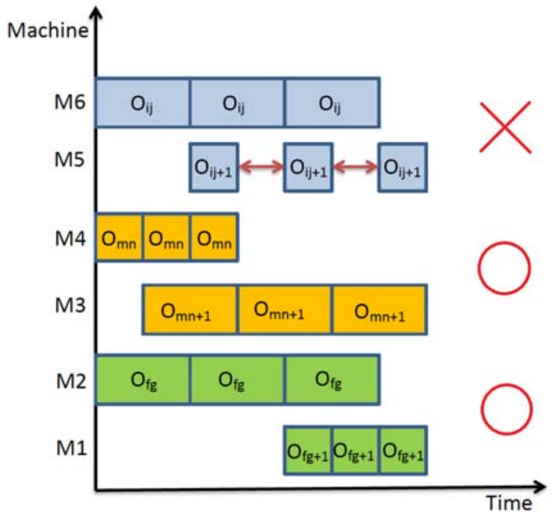

Figure 1. Three different cases of overlapping in operations.

To minimize the completion time, two situations described below should be considered:

A. $t_{kij} <= t_{kij+1}$

If the processing time of operation O_{ij+1} is greater than that of operation O_{ij}, the start time of operation O_{ij+1} should be no earlier than the time when the first sublot of operation O_{ij} is completed. The left part in Fig.2 is an example. The first sublot of O_{ij+1} starts when the first sublot of O_{ij} is completed, and the second and third sublots are operated right after the first sublot of O_{ij+1}.

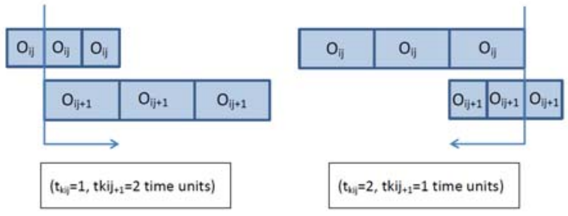
O_{ij}
O_{ij+1}
(t_{kij}=1, $tkij_{+1}$=2 time units)
(t_{kij}=2, $tkij_{+1}$=1 time units)

Figure 2. Consecutive operations with three sublots.

B. $t_{kij} > t_{kij+1}$

If the processing time of operation O_{ij+1} is smaller than that of operation O_{ij}, the last sublot of operation O_{ij+1} should be started after the entire operation O_{ij} is completed [11]. Otherwise idle time between sublots of an operation will occur. The right part in Fig.2 is an example. The third sublot of O_{ij+1} starts when the last sublot of O_{ij} is completed, and the first and second sublots are operated right before the last sublot of O_{ij+1}.

III. PROPOSED GA AND IMPROVEMENTS

A. Chromosome Encoding

In this paper, we use the same representation of chromosome as that in a previous research [12]. The representation includes an expression of machine selection (MS) and an expression of operation sequence (OS). In the MS part, an array of integer numbers is used in which each integer represents the index of an alternative machine that is able to process the operation. In the OS part, all operations of the same job are defined with their job index. All operations belong to the same job differ from each other based on the time the job index shows up. If a job index shows up *t* times in the OS part, it represents the *t*th operation of this job.

B. Chromosome Decoding

In a schedule of FJSP, global left shift means that some operations can be started earlier without delaying any other operations. When decoding a chromosome to a solution, only active schedules are considered. An active schedule means that no global left shift exists, in other words, no operation can be put into an idle interval earlier in the schedule while preserving feasibility. The most important property of active schedules is that they contain the optimal schedule [3].

To translate chromosomes into active schedules, priority-based decoding was used where we search the earliest available time interval for each operation.

C. Improved Population Initialization

Generating the initial population was considered separately for the MS part and the OS part of a chromosome. Assignment algorithm and random selection strategy [13] was used to form the MS part of a chromosome of the initial population. Meanwhile, the OS part of a chromosome of the initial population is obtained by a mix of three dispatching rules: random selection, most work remaining and most operations remaining [14].

The reason why we did not use the initialization method introduced by Demir and Isleyen [3] is that their methodology to initialize the OS part of a chromosome is time consuming. We think it is acceptable that we start from an initial population that is not the best but requires only a short time to generate.

D. Improved Selection Operator

D.a Original method

In the previous research, three-sized tournament selection operator was used [12].

D.b Improved method

Since the population size of each generation is constant, when using the original selection method to select *n* times among *n+x (n>>x)* individuals, each time one individual selected, there is a high possibility that individuals with high fitness are selected many times while individuals with low fitness are not selected at all. Such a trend becomes more obvious when the generation number increases since the number of individuals with high fitness is getting bigger. In later generations, the population will consist of many good but identical individuals.

We improve the selection operator as follows: we decrease the tournament selection size from 3 to 2, and in each tournament selection, the individual that has the higher fitness is added to the new population and removed from the parent population. By doing so, each individual can be selected only once. Thus the diversity of population is ensured.

E. *Improved Crossover Operator*

E.a Original method

In the previous research, the crossover operator is applied in MS and OS parts separately. A two-point crossover is applied in the MS part, and a precedence preserving order-based crossover (POX) [12] was used in the OS part. Fig.3 is an example for POX crossover. All 1's mean operations of job 1, 2's operations of job 2 and 3's operations of job 3. Job 2 is selected, so all operations of job 2 are copied from parent 1 to child 1 and from parent 2 to child 2, preserving their locations. Then all the other operations are copied from parent 1 to child 2 and from parent 2 to child 1, preserving their orders.

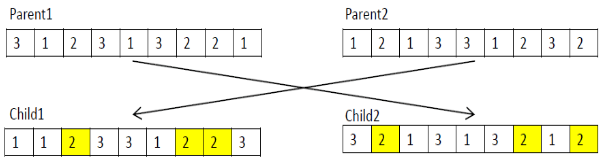

Figure 3. POX crossover for OS part.

E.b Improved method

Since the final schedule of a chromosome is determined by both the MS part and the OS part, if we change both of them in one generation, it is difficult to know which change leads to a better result and which does not. Thus we think it might be better to treat the MS part and the OS part as a whole in crossover.

Therefore, we apply POX crossover in only the OS part and let each of the produced children inherit the MS part from its parents. For example, if POX crossover in the OS part is performed as shown in Fig.3, then the change in the MS part would be performed as shown in Fig.4. In Fig.4, in each chromosome, the first three genes represent the three operations of job1, the next three genes represent the operations of job 2, and the last three genes represent the operations of job 3. The value of each gene means which machine is assigned for this operation. According to Fig.3, child1 preserves the location and the order of job2 in parent1 and the order of job1 and job3 in parent2. Therefore, in Fig.4, it should preserve the MS part of job2 in parent1 and the MS part of job1 and job3 in parent2. In Fig.4, the MS part of job2 in parent1, which is the fourth, fifth and sixth genes, are copied to child1 and the MS part of job1 and job3 in parent2, which is from the first to the third, and from the seventh to the ninth genes, are copied to child1.

This improvement takes the MS part and OS part as a whole, which helps us to determine whether the POX crossover in one generation is leading to a better result or not.

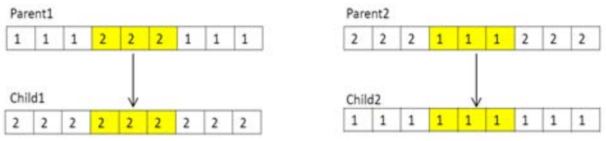

Figure 4. Inherit the MS part according to POX crossover in Fig.3.

F. *Improved Mutation Operator in the MS Part*

F.a Original method

In the previous research [12], the mutation operator is also performed on MS and OS parts separately. For the MS part, first a random operation is selected, and then the machine for the selected operation is changed to the machine that has the shortest processing time among the alternative machine set.

F.b Improved method

Given the fact that in FSJP, assigning an operation to the machine that has the shortest processing time may not always be the best choice for the whole schedule to achieve a smallest makespan, we try to improve the mutation in the MS part.

After selecting an operation randomly, we reassign the operation to an adjacent machine among the alternative machine set. That is, if the original assigned machine is machine1, then reassign to machine2, if the original machine is machine2, then reassign to machine3, and if machine 2 is the last machine in the alternative machine set, then reassign to machine1. It is beneficial for increasing diversity since it avoids the MS part to become the same for many individuals after many generations.

We propose a new mutation method in the MS part, which contains two steps: identifying the critical path and reassigning operations in it. This is because only moving operations in the critical path can affect the makespan, since the makespan is the finishing time of the last operation in the critical path.

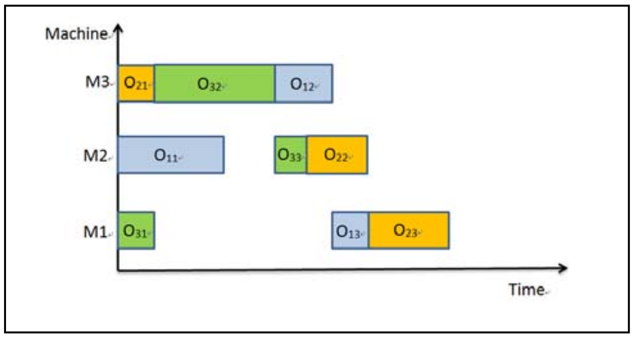

Figure 5. Finding the critical path.

When identifying the critical path, we start from the operation finished the latest. If several operations finished at the same time, we randomly choose one of them. Then there might be two paths to go on. One we call it the machine path, which goes to the previous operation on the same machine. The other we call it the job path, which goes to the previous operation of the same job. For each operation we choose one of the paths to go. In our method, we always choose the machine path first, and if it is not available, we choose the job path.

Take Fig.5 as an example. There is a schedule whose last operation is O_{23}. When we try to identify the critical path, we start from O_{23}. Since the start time of O_{23} is decided by O_{22} and O_{13}, we have two paths to go. The path going to O_{22} is called the job path, and the path going to O_{13} is called the machine path. According to our rule, we choose O_{13} to go. After that, the start time of O_{13} is decided by only O_{12}, so we go to O_{12} which is the job path. Then we go to O_{32}, which is the machine path. We then have two paths to continue, O_{21} is the machine path and O_{31} is the job path, so we go to O_{21} and find O_{21} starts at time 0, which means that the critical path ends.

There are several merits to choose the machine path first. The most important one is that this avoids the situation that some machines are crowded with operations while others are not. Therefore, to balance workload on machines so that the makespan could be reduced, it would be preferred that operations on the same machine are included as many as possible, because we can reassign them to other machines. At the same time, it is easier to determine whether a machine path is available than a job path. This is because for a machine path, we only need to check whether the finishing time of the previous operation is the start time of the current operation, but for a job path, due to the property of overlapping in operations, we have to calculate the expected finishing time based on the processing time of the previous operation and the current operation.

After identifying the critical path, we randomly select an operation in it, and then reassign the operation to an adjacent machine among the alternative machine set with the strategy we discussed before.

G. Improved Mutation Operator in the OS Part

G.a Original method

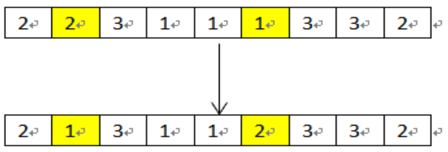

Figure 6. original mutation in the OS part.

In the previous research, two random genes are selected, the value of these selected genes swap with each other, as shown in Fig.6 [12].

G.b Improved method

Since swap might break the structure of the original schedule by changing which operation a gene represents in a job, we introduced the precedence preserving order-based shift proposed by Lee et al. [13]. The procedure is as follows: a random gene is selected, and then the gene is shifted to another position, while which operation this gene represents within its job is not changed.

As shown in the example in Fig.7, the second 2 is selected, which represents the second operation of job 2, then we can shift it only to the range from the first 2 to the third 2, which represent the first and the third operations of job 2 respectively. If the selected 2 is shifted out of this range, for example, to a position that is before the original first 2, then this 2 will become the first 2 and no longer represent the second operation of job 2. This might bring a huge influence to the structure of the original chromosome.

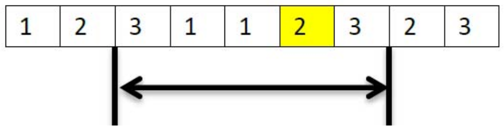

Figure 7. precedence preserving order-based shift.

H. Convergence and Diversity

Finally, we introduce a strategy that helps jumping out from a local optimum. That is if the global best result remains unchanged for a certain number of generations; we will randomly generate some new individuals and replace some worst individuals with these randomly generated individuals before crossover.

IV. Simulations and Analysis

In the simulations, we test our improved GA on a set of benchmark problems. The population size is 200 and the number of generations is 300. The simulation results are shown below.

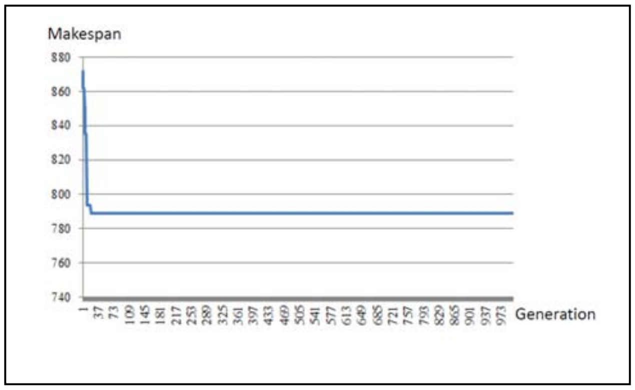

Figure 8. Result of the improved GA without strategy H.

Fig.8 is the result got by the improved GA without the strategy introduced H of section III. Fig.9 is the result got by the improved GA with strategy H. As shown by the two figures, the makespan can often reach a stable value within 50 generations, and by applying strategy H, better results could be obtained in late generations. The reason is that the new generated individuals may possess some chromosomes that contain different combinations of genes from those individuals in the current population. After adding them into the population, they can generate new offspring by crossover and mutation, and hence there is possibility that those offspring can outperform the original individuals.

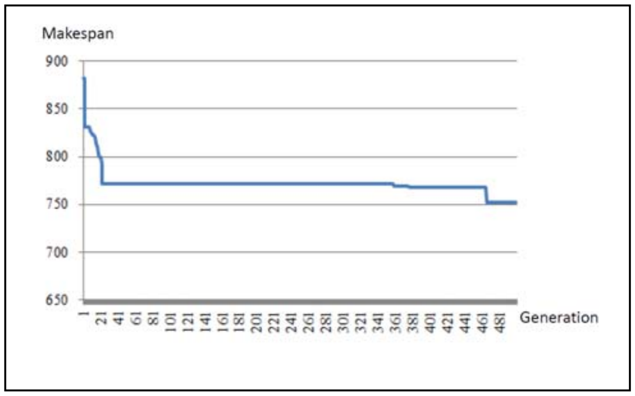

Figure 9. convergence figure for improved method.

From the results, we can find that the improved GA algorithm performs better on solving FJSP with overlapping in operations than the previous GA algorithm.

V. Conclusion and Future Work

In this paper, several improvements are proposed to improve the efficiency and diversity of a GA method in previous research. These improvements are based on the consideration to increase population diversity while preserving the convergence speed.

At present, all the improvements are considered based on the general FJSP. If we take the unique property of overlapping in operations into consideration, we might get better results.

TABLE I. Simulation results

Problem	**Size(i,j,k)**	**C_{max} without overlapping**	**AC_{max} by IGA**	**AC_{max} by PGA**	**MC_{max} by IGA**	**MC_{max} by PGA**
mt10c1	10.10.11	928	**784.7**	821.3	**764**	780
mt10cc	10.10.12	910	**764.8**	789	**748**	758
mt10x	10.10.11	918	**775.5**	801.8	748	**746**
mt10xx	10.10.12	918	**778**	805.1	**748**	779
mt10xxx	10.10.13	918	**771.9**	810.5	**742**	783
mt10xy	10.10.12	905	**761.3**	786.8	**733**	742
mt10xyz	10.10.13	847	**710.9**	745.9	**695**	703
setb4c9	15.10.11	914	**857.6**	886.9	**857**	865
setb4cc	15.10.12	909	**857**	870.6	**857**	**857**
setb4x	15.10.11	925	**856.2**	893.8	**846**	876
setb4xx	15.10.12	925	**858.7**	889.5	**846**	867
setb4xxx	15.10.13	925	**854.4**	887.1	**846**	864
setb4xy	15.10.12	916	**846.2**	873.2	**845**	850
setb4xyz	15.10.13	905	**844.7**	854.3	**842**	**842**
seti5c12	15.15.16	1174	**1038.8**	1067.2	**1032**	1042
seti5cc	15.15.17	1136	**977**	1033.4	**955**	1014
seti5x	15.15.16	1201	**1081.1**	1100	**1059**	1068
seti5xx	15.15.17	1199	**1077**	1099.8	**1053**	1068
seti5xxx	15.15.18	1197	**1073.6**	1099.7	**1050**	1077
seti5xy	15.15.17	1136	**976.4**	1018.9	**953**	990
seti5xyz	15.15.18	1125	**952.1**	1003.5	**940**	967

(IGA represents the improved GA algorithm with all improvements we presented; PGA represents the previous GA algorithm. AC_{max} represents the average makespan among 10 times; MC_{max} represents the minimum makespan among 10 times.)

Acknowledgment

The authors acknowledge the support of JSPS Grants-in-Aid for Scientific Research (KAKENHI) Grant Number 15K16296.

References

[1] Wang, X., L. Gao, C. Zhang, and X. Shao. 2010. “A Multi-objective Genetic Algorithm Based on Immune and Entropy Principle for Flexible Job-shop Scheduling Problem.” The International Journal of Advanced Manufacturing Technology 51: 757–767.

[2] Fantuhan, M., C. Defersha, and C. Mingyuan. 2009. “A Coarse-grain Parallel Genetic Algorithm for Flexible Job-shop Scheduling with Lot Streaming.” In International Conference on Computational Science and Engineering, Vancouver.

[3] Yunus Demir & Selçuk Kürşat İşleyen 2014 “An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations.” International Journal of Production Research, 52:13, 3905-3921

[4] Farugi, H., Y. Y. Babak, S. Hiresh, Z. Fyagh, and N. Foruzan. 2011. “Considering the Flexibility and Overlapping in Operation in Job Shop Scheduling Based on Meta-heuristic Algorithms.” Australian Journal of Basic and Applied Sciences 5 (11): 526–533

[5] H. Chen, J. Ihlow and C. Lehmann. 1999. “A Genetic Algorithm for Flexible Job-Shop Scheduling.” Proceedings of the 1999 IEEE International Conference on Robotics & Automation Detroit, Michigan May 1999

[6] F. Pezzellaa, G. Morgantia, and G. Ciaschettib. 2008. “A genetic algorithm for the Flexible Job-shop Scheduling Problem.” Computers & Operations Research 35 (2008) 3202 – 3212

[7] Garey, M. R., D. S. Johnson, and R. Sethi. 1976. “The Complexity of Flowshop and Jobshop Scheduling.” Mathematics of Operations Research 1: 117–129.

[8] Mastrolilli M, Gambardella LM. Effective neighbourhood functions for the flexible job shop problem. Journal of Scheduling 1996;3:3–20.

[9] Barnes JW, Chambers JB. Flexible Job Shop Scheduling by tabu search. Graduate program in operations research and industrial engineering. Technical Report ORP 9609, University of Texas, Austin; 1996.

[10] Chen H, IhlowJ, Lehmann C.Agenetic algorithm for flexible Job-shop scheduling. In: IEEE international conference on robotics and automation, Detroit; 1999. p. 1120–5.

[11] Torabi, S. A., B. Karimi, and S. M. T. Fatemi Ghomi. 2005. "The Common Cycle Economic Lot Scheduling in Flexible Job Shops: The Finite Horizon Case." International Journal of Production Economics 97: 52–65

[12] Zhang, G., L. Gao, and Y. Shi. 2011. "An Effective Genetic Algorithm for the Flexible Job-shop Scheduling Problem." Expert Systems with Applications 38: 3563–3573

[13] Lee, K. M., T. Yamakawa, and K. M. Lee. 1998. "A Genetic Algorithm for General Machine Scheduling Problems." International Journal of Knowledge-Based Electronic 2: 60–66

[14] Pezzellaa, F, Morgantia. G, Ciaschettib, G. 2007. "A Genetic Algorithm for the Flexible Job-shop Scheduling Problem." Computers & Operations Research 35 (2008) 3202 – 3212