J. Soc. Korea Ind. Syst. Eng Vol. 40, No. 2 : 111-120, June 2017
https://doi.org/10.11627/jkise.2017.40.2.111
ISSN : 2005-0461(print)
ISSN : 2287-7975(online)

An Improved Genetic Algorithm for Integrated Planning and Scheduling Algorithm Considering Tool Flexibility and Tool Constraints

Young-Nam Kim* · Chunghun Ha**†

*Department of computer science and engineering, Pohang university of Science and Technology
**School of Information & Computer Engineering, Hongik University

공구유연성과 공구관련제약을 고려한 통합공정일정계획을 위한 유전알고리즘

김영남* · 하정훈**†

*POSTECH 컴퓨터공학과
**홍익대학교 정보컴퓨터공학부 산업공학전공

This paper proposes an improved standard genetic algorithm (GA) of making a near optimal schedule for integrated process planning and scheduling problem (IPPS) considering tool flexibility and tool related constraints. Process planning involves the selection of operations and the allocation of resources. Scheduling, meanwhile, determines the sequence order in which operations are executed on each machine. Due to the high degree of complexity, traditionally, a sequential approach has been preferred, which determines process planning firstly and then performs scheduling independently based on the results. The two sub-problems, however, are complicatedly interrelated to each other, so the IPPS tend to solve the two problems simultaneously. Although many studies for IPPS have been conducted in the past, tool flexibility and capacity constraints are rarely considered. Various meta-heuristics, especially GA, have been applied for IPPS, but the performance is yet satisfactory. To improve solution quality against computation time in GA, we adopted three methods. First, we used a random circular queue during generation of an initial population. It can provide sufficient diversity of individuals at the beginning of GA. Second, we adopted an inferior selection to choose the parents for the crossover and mutation operations. It helps to maintain exploitation capability throughout the evolution process. Third, we employed a modification of the hybrid scheduling algorithm to decode the chromosome of the individual into a schedule, which can generate an active and non-delay schedule. The experimental results show that our proposed algorithm is superior to the current best evolutionary algorithms at most benchmark problems.

Keywords : Integrated Process Planning and Scheduling, Genetic Algorithm, Inferior Selection

Received 17 May 2017; Finally Revised 19 June 2017;
Accepted 20 June 2017
† Corresponding Author : Chunghun.ha@hongik.ac.kr

1. Introduction

Integrated process planning and scheduling problem (IPPS) is a mixture of process planning and scheduling. Process planning contains sub-problems of routing and loading. Routing focuses on determining which sequence to use among the replaceable operational sequences, and loading is a problem of determining which resources such as machines and tools are allocated to the operation for execution. A variety of resources and alternative sequences are available in manufacturing for the same manufacturing feature. On the other hand, scheduling is a problem of determining when the operations start and end on each machine, assuming that a process plan has been determined [6, 20].

In IPPS, several flexibilities and constraints have been considered. Process flexibility means that there are alternative operation sequences to be performed in a job. Sequence flexibility is a changeability of sequence order of operations. Machine and tool flexibilities imply that an operation can be performed on alternative machines with alternative tools and tool access direction, respectively. Process planning is related to the process, machine, and tool flexibilities, and scheduling targets the sequence flexibility. Each machine (or machining center) has a tool magazine that can mount multiple tools, thus can handle many kinds of operations in a machine. In practice, since the magazine capacity and the number of tools are limited, efficient decision-making is required under the limited manufacturing resources.

Due to the high degree of complexity, traditionally, a sequential approach has been preferred for IPPS, which determines process planning firstly and then performs scheduling independently based on the results. However, the two sub-problems are intricately interwoven. Thus, the optimal process plan does not result in the optimal schedule. Even the optimal process plan can lead to an infeasible schedule [19, 23]. It highlights the importance of the IPPS which solve both sub-problems simultaneously.

There is a myriad of alternative solutions in IPPS due to various flexibilities. It makes the problem complicated, but at the same time, it has the considerable potential to improve the performance of the system. Since IPPS is NP-hard [10], meta-heuristics have been mainly applied as optimization methods. According to Ausaf et al. [2] and Petrović et al. [21], GA, ant colony optimization (ACO), particle swarm optimization (PSO), and various memetic heuristics combined with these have frequently been applied for IPPS [12]. Among them, GA is the most preferred. With IPPS having a huge solution space, the superior exploitation capability of GA is advantageous to improve the solution quality.

There are abundant studies to solve IPPS using GA. Morad and Zalzala [18] proposed a simple GA for IPPS considering machine flexibility only. To deal with various flexibilities such as process, sequence, and machine in IPPS, Kim et al. [11] introduced a symbiotic evolutionary algorithm that mimics the process of coevolution of various species. Kim et al. [10] improved their previous study by proposing a multi-layered symbiotic evolutionary algorithm for IPPS, which considers tool flexibility, tool magazine capacity constraint, and tool capacity constraint additionally compared to previous IPPS. Shao et al. [22] proposed a GA that establishes an optimal plan reflecting the real-time status of the shop floor. Liu et al. [17] developed an adaptive annealing genetic algorithm for IPPS in a job shop, which adopts Boltzmann probability selection using adaptive mutation probability and simulated annealing to avoid premature convergence. Li et al. [14] presented a mathematical programming modeling for IPPS and optimized it by an evolutionary algorithm. Li et al. [15] also proposed a hybrid algorithm that uses the GA as the main algorithm and applies the Tabu search to further improve the solution. Li et al. [13] proposed a GA that improves premature convergence by a learning operator that perform a crossover operation between the currently selected solution and the current best solution or the generation's best solution. Lian et al. [16] applied the imperialist competitive algorithm [1] to enhance exploitation, which evolves empires consisting of an imperialist and colonies in parallel. Zhang and Wong [24] proposed a GA with object-coding representation, inferior selection, job-based precedence preservative crossover, shifting gene and shifting machine mutations, and crowding replacement scheme. Cinar et al. [7] proposed a constructive GA based on the Giffler-Thompson algorithm [8] for flexible job shop scheduling, in which a priority-based representation that consists of the priority of operations is used.

Although many studies using GA for IPPS have been conducted in the past, tool flexibility and capacity constraints are not considered except for Kim et al. [10]. Also, most studies use the problem set of Kim et al. [11] as a benchmark to evaluate the performance of their algorithms. This paper, which considers tool flexibility and capacity constraints, uses the new problem set of Kim et al. [10] as a benchmark. In this respect, this study is different from the existing studies.

In this paper, we aim at developing an improved genetic

algorithm for IPPS with tool flexibility and tool related constraints. We adopted a traditional genetic algorithm procedure, but we employed three methods to improve solution quality and computation time. The first is the initial population generation using a random circular queue. It binds the population size, which decreases computation time critically. Second, we adopted an inferior selection to maintain exploitation capability in the evolution process. Third, we employed a modification of the hybrid scheduling algorithm [3, 10] to decode the chromosome of the individual into a schedule.

We organized this paper as follows. Section 2 explains IPPS in detail after the description on flexibilities for the flexible manufacturing system and network representation. The proposed genetic algorithm is introduced in Section 3. Section 4 is prepared for describing the experimental parameters and results. Finally, we conclude and discuss our research in Section 5.

2. IPPS Problem

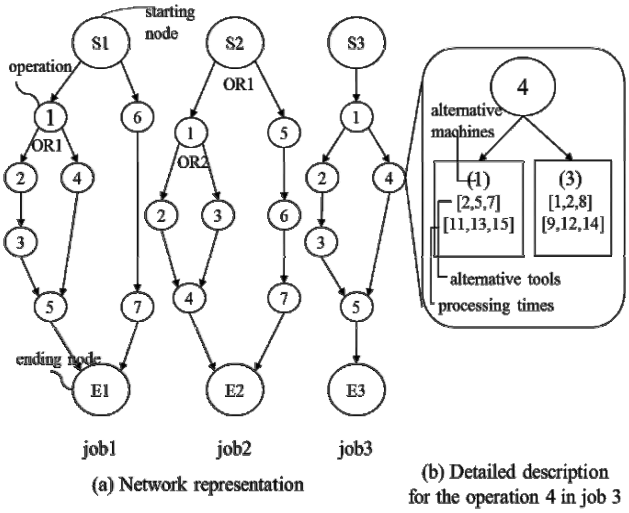
A simple description of the IPPS problem is as follows [5].

(1) There is a set of n jobs to be processed on m machines, $M = \{M_1, M_2, \cdots, M_m\}$
(2) The job i consists of a sequence of n_i operations, $(o_{i1}, o_{i2}, \cdots, o_{in_i})$
(3) The set of t tools is noted as $T = \{T_1, T_2, \cdots, T_t\}$
(4) Each operation must be assigned a machine and a tool for execution. Let M_k and T_l be assigned on the o_{ij}. Then, processing time of o_{ij} is $p_{ij,kl}$.

We assume following operating conditions:

(a) All machines are available at the starting time of scheduling.
(b) All jobs are prepared at the starting time of scheduling.
(c) A machine cannot process multiple operations simultaneously.
(d) Each operation must be processed at once, that is, it cannot be separated.
(e) If it is not defined in the network representation, there are no precedence constraints among the operations.
(f) Pre-emption of an operation is not allowed.
(g) Processing time includes transportation time and setup time.

We will illustrate the problem in detail using the network representation by Ho and Moodie [9]. <Figure 1> shows a



<Figure 1> An Example of IPPS with Three Jobs [10]

simple example of an IPPS. There are a total of three jobs, job 1, job 2, and job 3, each of which is a network of operation nodes. The node number is the identifier of an operation in the job, that is, the operation 1 of job 1 and the operation 1 of job 2 may be different. Each job network has two dummy nodes indicating the start (S1, S2, and S3) and the end of the job (E1, E2, and E3). Arrowed lines indicate precedence constraints. The operations with the tail of the arrowed line are the immediate predecessors of the operation indicated by the head of the arrow. That is, the head operation must be processed after the tail operation is completed. For example, operation 1 of job 1 must be completed before operation 2 of the job is executed. They cannot be processed simultaneously, and succeeding operations cannot be pre-processed.

In the network representation, OR notation implies process flexibility, only one branch can be selected for processing. If a schedule does not satisfy any precedence constraint, it becomes infeasible. In the network representation, the branching node without OR notation such as operation 1 in job 3 denotes an AND node that all operations belonging to the branch must be processed to complete the job. For instance, job 1 consists of 9 nodes, 1 AND node (S1), 1 OR node (operation 1), and seven ordinary nodes (operation 2 to 7 and E1). The S1 node has two process routes (operation 1 to 5 and 6-7) and both routes have to be processed because it is an AND node. The operation 1, on the other hands, is an OR node which has two process routes, (2-3) and (4). They can produce the same manufacturing features. This means that we can choose either one, and the unselected one is not taken into account when creating a schedule. For example, If (4) is selected, then operation 2 and 3 become

dummy operations. In the opposite case, operation 4 becomes a dummy operation.

Finally, we have to determine which machines and tools will process the operations. In <Figure 1(b)>, it shows the possible combination of alternative machines and tools which process the operation 4 of job 3. It also presents the processing times depending on the decisions. For example, if operation 4 is run on machine 3 using tool 8, it will take 14-time units to complete.

3. Proposed Genetic Algorithm

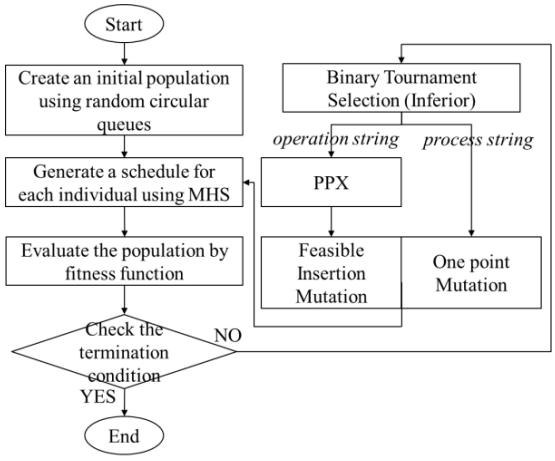
3.1 Overall GA Procedure

We adopt a traditional genetic algorithm procedure. <Figure 2> shows the procedure steps and the techniques used in this paper. At the initial population creation step, a certain number of individuals are generated using a circular queue that uniformly allocates the machine and tool to be processed on the operations in the individual. The next sub-section will illustrate the detailed explanation on it. Next, all individuals in the population are evaluated by a fitness function. Since the objective of our IPPS is to minimize the makespan, which is the completion time of all operations, decoding (scheduling in IPPS) has to be performed on each individual. We use a modified hybrid scheduling algorithm (MHS) that is a modification of Bierwirth and Mattfeld [3, 10]. Then, the GA checks the termination condition that is a fixed number of generations to be evolved. If the condition is passed, the algorithm performs the selection, crossover, and mutation operations sequentially on the population. As a result, the offspring replaces the parents, and the new generation starts. If the termination condition meets, the algorithm stops and the current best solution becomes the final solution.

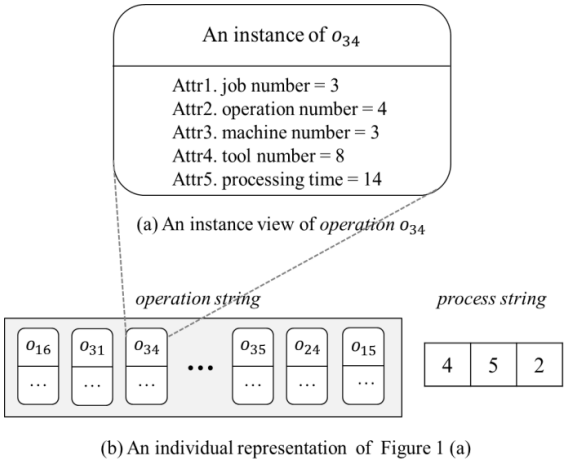
3.2 Individual Representation

Due to the high flexibility, a simply structured chromosome representation for IPPS is not allowed. The traditional representation for IPPS [10, 15, 16, 17, 21], constructs separate representations for each required information. For example, the information on the machine allocation is stored in individual W, the information on the tool allocation is stored in individual X, the information on the execution order is stored in individual Y, and the information on the selected process route is stored in individual Z. Each of these does not represent a complete manufacturing plan. Only when the individuals W, X, Y, and Z are combined, a meaningful manufacturing plan can be obtained and evaluated. Moreover, the multiple types of individuals require multiple populations to be evolved, which results in a complicated GA procedure and a long calculation time.

To overcome these weaknesses and create a manageable representation for IPPS, we propose a permutation representation composed of entities that include all required information. To represent the integrated manufacturing plan as a single object, we created a new data structure, namely *operation*. Hereafter, the name of the new data structure will be italicized to distinguish those from the existing terms. The *operation* is composed of the following five attributes; job number, operation number, machine number, tool number, and processing time. For example, In <Figure 1(b)>, the operation 4 of the job 3, i.e., o_{34}, can be instantiated as an operation, as shown in <Figure 3(a)>, in which all attributes are assigned for scheduling.



<Figure 2> Steps of GA and Used Methods



<Figure 3> The Representation of *Operation* and a Sequence

<Figure 3(b)> shows a sample sequence of *operations* called as *operation string* and a sample array of process routes called as *process string*. Although the example shows only several selected *operations*, the complete individual must contain all the *operations* of whole jobs, including dummy operations that are not actually performed by OR relations. The *operation string* alone is not enough to make a complete individual because there is no information about which alternative process route is to be performed. The *process string* provides such information. It is an array of the numbers of the starting operations of the alternative sequences at each OR relation. The length of the *process string* is the sum of the number of OR relations in all jobs. For example, In <Figure 1(a)>, job 1 has just one OR relation having 2 alternative process routes ({2-3} and {4}), job 2 has 2 OR relations (the first one branches off to {1-2-3-4} and {5-6-7} and the second one branches off to {2} and {3}), and job 3 does not have any OR relations. Thus, the length of the *process string* is 3 and can be denoted as an array, e.g., [4, 5, 2] and [2, 1, 3].

3.3 Modified Hybrid Scheduling Algorithm

The individual composed of an *operation string* and a process string is not a final solution for IPPS. The individual must be decoded into a schedule with the information on the chromosome. An operation is schedulable only if its predecessors have been scheduled already. There are four types of schedule : semi-active, active, non-delay, and hybrid. It is well-known that the optimal schedule is an active schedule. Bierwirth and Mattfeld [3, 10] proposed a hybrid scheduling algorithm by modifying Giffler and Thompson [8], which can generate an active and non-delay schedule. In this paper, we adopted the scheduling algorithm, but partially modified it to consider dummy nodes. The detailed procedure is as follows and we follow notations of Kim et al. [10].

n : the number of jobs
o_{ij} : the operation j of the job i
λ_{ij} : the earliest starting time of the operation o_{ij}
p_{ij} : the processing time of the operation o_{ij}
$\tau_{ij} = \lambda_{ij} + p_{ij}$: the earliest completion time of the operation o_{ij}.

(1) Set j as 0 for all dummy operations.
(2) Construct a set A of all possible starting operations except dummy operations. For example, in <Figure 1>, $A = \{o_{11}, o_{16}, o_{25}, o_{31}\}$ if the *process string* is [2, 3, 5].
(3) Find $\tau^* = \min\{\tau_{ij} \mid o_{ij} \in A\}$ and let the machine of the operation with τ^* as m^*.
(4) Construct a set B = $\{o_{ij} \mid o_{ij} \in A$ and o_{ij} runs on $m^*\}$ and calculate $\lambda^* = \min\{\lambda_{ij} \mid o_{ij} \in B\}$.
(5) Construct a set C : = $\{o_{ij} \in B \mid \lambda_{ij} \le \theta\tau^* + (1-\theta)\lambda^*, 0 \le \theta \le 1\}$.
(6) Select the operation o_{ij}^* at the left-most from C and delete it from A.
(7) Append the operation o_{ij}^* on the schedule and calculate its starting and completion time.
(8) All the successors of operation o_{ij}^* which are not dummy nodes and their all predecessors have been scheduled.
(9) If A is not empty, go to (3). Otherwise, stop.

The design parameter θ determines the type of schedule, i.e., zero yields a non-delay schedule and one yields an active schedule. In our algorithm, we set θ as 0.5 which is known to be the best empirically by Kim et al. [10].

The MHS consumes much computing resources due to frequent search processes. To overcome it, we developed an *index array* which is a 2-dimentional array of indices of *operation*. Through this, we can find out where the specific operation is in the sequence without any search algorithm. For example, in <Figure 3(b)>, the index of o_{34} is 3, so the third row and fourth column component of the index array is 3. That is, the *index array* is a lagged array which row number corresponds to the job number and column number corresponds to the operation number. Of course, the search process can be used without the *index array*, but the time complexity increases from $O(1)$ to $O(n)$. It makes stark difference for such a complex problem.

3.4 Fitness Function

According to Petrovic et al. [21], makespan is the most popular objective function for IPPS. If the IPPS pursues multi-objectives, tardiness, workload, and machine utilization are considered further. In this paper, we choose the makespan as a single objective. Our IPPS problem considers the constraints on tool magazine capacity and tool capacity. Thus, although all precedence constraints are satisfied, it may result in an infeasible solution if the number of used tool types on a machine or the number of used tools on all machines exceeds the tool and tool magazine capacities, respectively. However, it is hard to check and repair the violation of those

constraints during genetic operations. It consumes a long computation time and decreases exploitation capability. Therefore, we imposes an amount of penalty [10] for the violation of constraints as follows :

$$\text{penalty} = c_1 \sum_t TP(t)^{\alpha} + c_2 \sum_m MP(m)^{\beta}$$

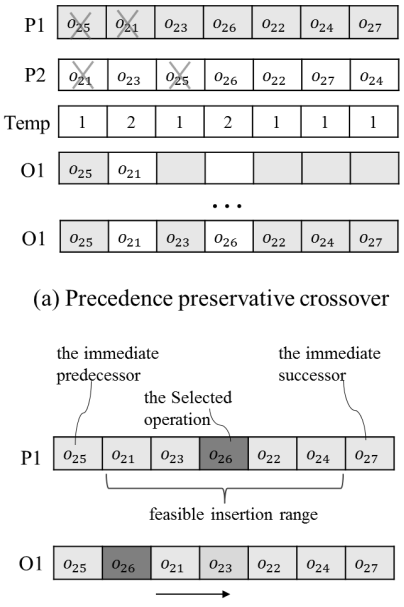
Here, tool penalty, $TP(t)$, denotes the excessive number of the specified usage of tool t. $MP(m)$ denotes the excessive number of the tool magazine capacity for machine m. Where c_1, c_1, α and β are parameters. At the end, the fitness of an individual becomes fitness = 1/(makespan+penalty).

3.5 Initial Population

IPPS with a large solution space should maintain a sufficient exploitation capability to derive an excellent optimal solution and to prevent premature convergence of the GA. The easiest way to solve this is to construct a large population. However, it causes a long calculation time and makes the GA's efficiency worse. While generating an individual that belongs to the initial population, all attributes of *operation*s must be assigned. Otherwise, a fitness evaluation of the individual is impossible. A typical random assignment for the attributes limits the solution space if the population size is small. For this reason, we propose a method that assigns the attributes evenly using circular queues when constructing the initial population. A circular queue consists of all combinations of possible resource assignments for each operation. For example, if o_{34} is processed on machine 1 using tool 2, the combination of machine and tool can be denoted as a tuple, (1, 2). In <Figure 1(b)>, all the possible combinations can be listed as follows; (1, 2), (1, 5), (1, 7), (3, 1), (3, 2), (3, 8). The circular queue corresponding to o_{34} are generated by shuffling the list arbitrarily. While generating an individual, one element is taking out from the circular queue of the *operation* and the machine, tool, and subsequent processing time of it is assigned into the *operation*. Then, the element is reinserted into the tail of the circular queue. By doing this, all resources can be distributed evenly. This helps to maintain a reasonable size of population without serious loss of solution quality. An *operation string* including dummy operations is generated by applying this process iteratively. Finally, the *operation string* is shuffled randomly. Since the proposed GA applies a sequence independent hybrid scheduling, precedence constraints need not be satisfied. The *process string* is also generated by arbitrarily selecting one of the starting operations of the alternative process routes on every OR relations.

3.6 Selection, Crossover, and Mutation

In GA, the selection operation is to select parents to perform the crossover and mutation operations. A typical binary tournament selection is to randomly pick up two individuals in the population and select the one with a higher fitness. According to GA's philosophy of inheriting superior genes, the superior selection is appropriate. However, some recent studies on IPPS [24, 25] show that the inferior selection which selects an individual with lower fitness yields better results than the superior selection. It is because the inferior selection maintains exploitation capability and prevents premature convergence in IPPS with high complexity. We already applied the circular queue for the generation of the initial population for this purpose, but the inferior selection plays a role in supplementing it.



(a) Precedence preservative crossover

(b) Feasible insertion

<Figure 4> PPX and Feasible Insertion

Since the chromosome is combined by the *operation string* and the *process string*, a crossover and a mutation operation should be performed respectively. For the *operation string*, we adopt the well-known precedence preservative crossover (PPX) [3, 4]. The PPX is performed as follows.

(1) Generate an empty child *operation string* that has the same length as the parents' *operation string*.
(2) Select 1 or 2 randomly.
(3) If the number is 1 (2), copy the operation at the beginning of P1 (P2) and append it to the child.
(4) Delete the operation with the same job number and operation number from both P1 and P2, respectively.
(5) If any element of parents remains, go back to step (2). Otherwise, terminate the crossover operation.

As for mutation, we used a feasible insertion for the *operation string*. A selected operation is inserted at any position between the immediate preceding and succeeding operation.

In *process string*, we do not perform any crossover operation because there was no significant difference in preliminary tests. However, for mutation, we employed one point mutation for the *process string*, in which a randomly selected element is replaced arbitrarily by one of the alternatives.

4. Experimental Results

A series of experiments were performed on the benchmark problems proposed by Kim et al. [10]. <Table 1> shows the 31 problems with a total of 18 jobs (parts). <Table 2> shows the information of the tool magazine capacity on each machine (Mcapa.), tool capacity (Tcapa.), and the required number of slots to mount each tool on a tool magazine (Req. slots).

All experiments were performed 10 times for each problem. The population size was set to be 300 regardless of the problem. The termination condition is the number of generation, which is 300 generations for all problems. Also, we assigned the crossover rate as 0.6 and the mutation rate as 0.05 for all problems. For the calculation of fitness, values of 10, 10, 0.5 and 0.5 were used for c_1, c_2, α, and β similar to Kim et al. [10], respectively. The reason for it is to compare fairly the performance of our proposed algorithm. The proposed GA was implemented in Java and has been performed on an Intel i7-5700HQ 2.7 GHz CPU.

<Table 3> and <Figure 5> present the comparison of the results of the proposed algorithm with previous evolutionary algorithms. Here, HEA represents the hierarchical evolutionary algorithm [10] and AMSEA denotes the asymmetric multileveled symbiotic evolutionary algorithm [10]. In the table, the shortest makespans for each problem among the algorithms are shown in bold. The improved rate indicates the degree of improvement compared with the best makespan among HEA and AMSEA. The gray cell denotes improved solution.

<Table 1> Benchmark Problems [10]

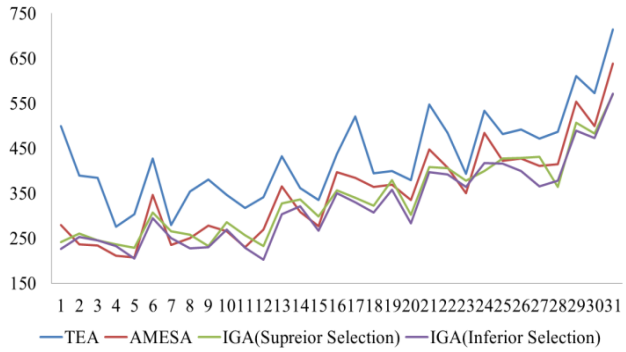
Problem	Job number
Prm01	1, 2, 3, 10, 11, 12
Prm02	4, 5, 6, 13, 14, 15
Prm03	7, 8, 9, 16, 17, 18
Prm04	1, 4, 7, 10, 13, 16
Prm05	2, 5, 8, 11, 14, 17
Prm06	3, 6, 9, 12, 15, 18
Prm07	1, 4, 8, 12, 15, 17
Prm08	2, 6, 7, 10, 14, 18
Prm09	3, 5, 9, 11, 13, 16
Prm10	4, 5, 6, 10, 11, 12
Prm11	7, 8, 9, 13, 14, 15
Prm12	1, 2, 3, 16, 17, 18
Prm13	1, 2, 3, 5, 6, 10, 11, 12, 15
Prm14	4, 7, 8, 9, 13, 14, 16, 17, 18
Prm15	1, 4, 5, 7, 8, 10, 13, 14, 16
Prm16	2, 3, 6, 9, 11, 12, 15, 17, 18
Prm17	1, 2, 4, 7, 8, 12, 15, 17, 18
Prm18	3, 5, 6, 9, 10, 11, 13, 14, 16
Prm19	4, 5, 6, 7, 8, 9, 10, 11, 12
Prm20	1, 2, 3, 13, 14, 15, 16, 17, 18
Prm21	1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 14, 15
Prm22	4, 5, 6, 7, 8, 9, 13, 14, 15, 16, 17, 18
Prm23	1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17
Prm24	2, 3, 5, 6, 8, 9, 11, 12, 14, 15, 17, 18
Prm25	1, 2, 4, 6, 7, 8, 10, 12, 14, 15, 17, 18
Prm26	2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 16, 18
Prm27	4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
Prm28	1, 2, 3, 7, 8, 9, 13, 14, 15, 16, 17, 18
Prm29	2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18
Prm30	1, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18
Prm31	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18

<Table 2> Tool Usage and Tool Magazine Constraint [10]

Tool magazine capacity		Tool capacity					
m	Mcapa. [slots]	*t*	Tcapa. [EA]	Req. slots	*t*	Tcapa. [EA]	Req. slots
1	28	1	6	1	11	7	1
2	31	2	7	2	12	10	2
3	38	3	6	2	13	9	3
4	28	4	6	2	14	8	2
5	36	5	10	1	15	6	2
6	37	6	6	2	16	10	2
7	29	7	10	3	17	9	1
8	28	8	9	2	18	6	2
9	29	9	7	1	19	8	2
10	26	10	5	2	20	7	3

<Table 3> Makespan Comparison of Various Evolutionary Algorithms

Problem	HEA [10]			AMESA [10]			IGA (Superior Selection)			IGA (Inferior Selection)			
	best	mean	s.d.	best	average	std	best	average	std	best	average	std	improved rate [%]
Prm01	500	506.7	11.7	280	292.2	9.8	242	272.6	16.7	**226**	253.7	12.8	6.6
Prm02	389	398.1	15.1	**237**	257.3	9.4	261	284.2	15	253	268.9	13.7	-6.8
Prm03	385	391	8	**234**	246.6	7.1	245	279.7	20.9	246	262.9	10	-5.1
Prm04	276	288.8	19.6	**211**	219.1	5	237	254.4	16.1	233	242	10.6	-10.4
Prm05	304	314.7	22.7	207	214.2	5.6	229	250.7	9.5	**205**	233.7	17.3	1.0
Prm06	428	429.4	2.5	346	365.5	13.2	308	326.2	10.4	**295**	310.3	12.7	4.2
Prm07	279	295.4	15.7	**235**	263.4	12.8	266	293.2	13.8	250	276.7	17.7	-6.4
Prm08	354	357.5	5.3	250	258.8	6.6	258	280.2	15.9	**228**	253	14.5	8.8
Prm09	381	382	2.1	278	291.9	8.9	233	268	21.9	**230**	258	14.2	1.3
Prm10	347	349.1	4.2	**266**	284.1	7.9	286	301.3	9	270	289.3	15.6	-1.5
Prm11	317	319.5	4.2	230	243.3	12.1	257	273.4	10.8	**229**	247.7	14.6	0.4
Prm12	342	349	9.9	269	286.3	12.9	233	254.2	12.3	**203**	226.8	13.3	12.9
Prm13	432	441.8	8.4	365	379.6	10.5	328	353.6	14	**304**	341.4	15.8	7.3
Prm14	362	394.2	21	**309**	335.1	16.5	337	351.5	9.9	321	351.8	20.6	-3.9
Prm15	335	341.5	5.7	277	283.3	5.9	298	318.1	15.9	**267**	296	17.4	3.6
Prm16	438	480.2	21.3	397	421.9	14.5	357	384.4	18.8	**351**	370.8	13.3	1.7
Prm17	521	522.8	3	385	399.9	8.4	340	371	14.6	**330**	351	13.6	2.9
Prm18	395	422.2	16.2	364	370.7	5.7	323	346	13.7	**307**	330.7	18	5.0
Prm19	400	428.6	21.9	370	383.2	7.5	380	411.3	20.5	**358**	387.5	16.4	3.2
Prm20	380	384.7	4.1	335	343.1	7.3	302	317.5	17.4	**284**	298.5	11.8	6.0
Prm21	548	549.6	2.6	448	465.1	9.8	408	433	20.9	**397**	421.4	16.1	2.7
Prm22	485	514.1	17.5	407	426.5	12.4	406	444	21.8	**392**	429.7	22.4	3.4
Prm23	394	443.7	24.4	351	362.9	8.3	378	395.4	10.9	364	386.5	14.9	-3.7
Prm24	534	587.5	27.3	484	500.2	7.4	400	449	21.4	**418**	459.7	18.2	-4.5
Prm25	482	531.3	31.2	422	448.7	13.7	427	453.2	20.4	**416**	439.8	14.3	1.4
Prm26	492	529.5	28.6	428	455.1	13.9	429	444.6	12.5	**400**	428.2	17.4	6.5
Prm27	472	508.1	32	411	435.6	12.5	431	454.2	15.7	**365**	424.4	23.5	11.2
Prm28	487	518.2	30.7	415	440.3	14.4	364	410	21	**378**	399.7	15.7	-3.8
Prm29	611	638.1	21.7	554	580.2	13.7	507	537.3	20.5	**489**	521.8	16.5	3.6
Prm30	573	602.1	11.6	499	526.6	13	483	524.8	20.3	**473**	522.2	22.6	2.1
Prm31	714	753	32.7	639	669.2	19.2	570	608.7	23.2	572	595.3	12.1	-0.4



<Figure 5> Best Makespan Comparison

The results show that our proposed algorithm (IGA), particularly inferior selection mechanism, is much better than those of HEA and AMSEA. Although direct comparison of computation time is impossible, the results are satisfactory under the given population size of 300 individuals and the termination condition of 300 generations.

5. Conclusion

In this paper, we proposed an improved genetic algorithm

for IPPS with tool flexibility and tool related constraints. Experiments on the benchmark problems proved that the proposed GA is superior to the existing best algorithm. In a practical manufacturing environment, if the subsequent operation uses another machine, the loading and unloading times of the workpiece is essential. Additional setup time should also be considered if the orientation of the tool or workpiece changes on the same machine. The IPPS in this paper assumed that these times are included in the processing time. Future studies will cover more realistic IPPS by considering such times.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A1A 01060391).

References

[1] Atashpaz-Gargari, E. and Lucas, C., Imperialist competitive algorithm : An algorithm for optimization inspired by imperialistic competition, *2007 IEEE Congress on Evolutionary Computation, CEC 2007*, 2007, pp. 4661-4667.

[2] Ausaf, M.F., Li, X., and Gao, L., Optimization Algorithms for Integrated Process Planning and Scheduling Problem-A Survey, 2014, pp. 5278-5283.

[3] Bierwirth, C. and Mattfeld, D.C., Production scheduling and rescheduling with genetic algorithms, *Evolutionary computation*, 1999, Vol. 7, No. 1, pp. 1-17.

[4] Blanton Jr, J.L. and Wainwright, R.L., Multiple vehicle routing with time and capacity constraint using genetic algorithms, *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1993, pp. 452-459.

[5] Chaudhry, I.A. and Khan, A.A., A research survey : Review of flexible job shop scheduling techniques, *International Transactions in Operational Research*, 2016, Vol. 23, No. 3, pp. 551-591.

[6] Cho, S., Lee, H., and Kim, S., A Study on Dynamic Scheduling in Flexible Manufacturing System Environment, *Journal of the Society of Korea Industrial and Systems Engineering*, 2004, Vol. 27, No. 2, pp. 17-23.

[7] Cinar, D., Oliveira, J.A., Topcu, Y.I., and Pardalos, P.M., A priority-based genetic algorithm for a flexible job shop scheduling problem, *Journal of Industrial and Management Optimization*, 2016, Vol. 12, No. 4, pp. 1391-1415.

[8] Giffler, B. and Thompson, G.L., Algorithms for Solving Production-Scheduling Problems, *Operations Research*, 1960, Vol. 8, No. 4, pp. 487-503.

[9] Ho, Y.-C. and Moodie, C.L., Solving cell formation problems in a manufacturing environment with flexible processing and routeing capabilities, *International Journal of Production Research*, 1996, Vol. 34, No. 10, pp. 2901-2923.

[10] Kim, Y.K., Kim, J.Y., and Shin, K.S., An asymmetric multileveled symbiotic evolutionary algorithm for integrated FMS scheduling, *Journal of Intelligent Manufacturing*, 2007, Vol. 18, No. 6, pp. 631-645.

[11] Kim, Y.K., Park, K., and Ko, J., A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling, *Computers & Operations Research*, 2003, Vol. 30, No. 8, pp. 1151-1171.

[12] Lee, D., Applying tabu search to multiprocessor task scheduling problem with precedence relations, *Journal of the Society of Korea Industrial and Systems Engineering*, 2004, Vol. 27, No. 4, pp. 1-6.

[13] Li, X., Gao, L., and Shao, X., An active learning genetic algorithm for integrated process planning and scheduling, *Expert Systems with Applications*, 2012, Vol. 39, No. 8, pp. 6683-6691.

[14] Li, X., Gao, L., Shao, X., Zhang, C., and Wang, C., Mathematical modeling and evolutionary algorithm-based approach for integrated process planning and scheduling, *Computers & Operations Research*, 2010, Vol. 37, No. 4, pp. 656-667.

[15] Li, X., Shao, X., Gao, L., and Qian, W., An effective hybrid algorithm for integrated process planning and scheduling, *International Journal of Production Economics*, 2010, Vol. 126, No. 2, pp. 289-298.

[16] Lian, K., Zhang, C., Gao, L., and Li, X., Integrated process planning and scheduling using an imperialist competitive algorithm, *International Journal of Production Research*, 2012, Vol. 5015, No. 15, pp. 4326-4343.

[17] Liu, M., Sun, Z.J., Yan, J.W., and Kang, J.S., An adaptive annealing genetic algorithm for the job-shop planning and scheduling problem, *Expert Systems with Applications*, 2011, Vol. 38, No. 8, pp. 9248-9255.

[18] MORAD, N. and A. Zalzala, Genetic algorithms in integrated process planning and scheduling, *Journal of*

Intelligent Manufacturing, 1999, Vol. 10, No. 2, pp. 169-179.

[19] Nasr, N. and Elsayed, E.A., Job shop scheduling with alternative machines, *International Journal of Production Research*, 1990, Vol. 28, No. 9, pp. 1595-1609.

[20] Park, B.J. and Lee, S.W., Job Shop Scheduling with Evolutionary Algorithms, *Journal of The Korean Institute of Plant Engineering*, 2000, Vol. 5, No. 2, pp. 95-102.

[21] Petrovic, M., Vukovic, N., Mitic, M., and Miljkovic, Z., Integration of process planning and scheduling using chaotic particle swarm optimization algorithm, *Expert Systems with Applications*, 2016, Vol. 64, pp. 569-588.

[22] Shao, X.Y., Li, X.Y., Gao, L., and Zhang, C.Y., Integration of process planning and scheduling-A modified genetic algorithm-based approach, *Journal of Computers & Operations Research*, 2009, Vol. 36, No. 6, pp. 2082-2096.

[23] Thomalla, C.S., Job shop scheduling with alternative process plans, *International Journal of Production Economics*, 2001, Vol. 74, No. 1-3, pp. 125-134.

[24] Zhang, L. and Wong, T.N., An object-coding genetic algorithm for integrated process planning and scheduling, *European Journal of Operational Research*, 2015, Vol. 244, No. 2, pp. 434-444.

[25] Zhang, S., Yu, Z., Zhang, W., Yu, D., and Xu, Y., An Extended Genetic Algorithm for Distributed Integration of Fuzzy Process Planning and Scheduling, *Mathematical Problems in Engineering*, 2016, Vol. 2016, pp. 1-13.

ORCID

Young-Nam Kim | http://orcid.org/0000-0002-0762-6896
Chunghun Ha | http://orcid.org/0000-0002-4222-2555