Iranian Journal of Management Studies (IJMS) *http://ijms.ut.ac.ir/*
Vol. 12, No. 2, Spring2019 *Print ISSN: 2008-7055*
pp. 307-333 *Online ISSN: 2345-3745*
DOI: 10.22059/ijms.2019.255363.673086

Integrated Process Planning and Active Scheduling in a Supply Chain-A Learnable Architecture Approach

Esmaeel Moradi[1], Ashkan Ayough[2], Mostafa Zandieh[2]*

1. School of Industrial Engineering and Management, Oklahoma State University, Stillwater, USA
2. Department of Industrial Management, Management and Accounting Faculty, Shahid Beheshti University, Tehran, Iran

(Received: August 19, 2018; Revised: March 11, 2019; Accepted: April 7, 2019)

Abstract

Through the lens of supply chain management, integrating process planning decisions and scheduling plans becomes an issue of great challenge and importance. Dealing with the problem paves the way to devising operation schedules with minimum makespan; considering the flexible process sequences, it can be viewed as a fundamental tool for achieving the scheme, too. To deal with this integration, the modeling approach to problem with MIP structure is common in the literature. These models take precedence constraints into consideration to select machines and to determine sequences. In order to obtain viable sequences, we employed a proposed transformation matrix (TM). We also took advantage of an evolutionary search, called Learnable genetic Architecture (LEGA). Based on LEGA, we developed an integrated process planning and scheduling learnable genetic algorithm (IPPSLEGA). Our approach was evaluated with problems with various sizes. The experimental results show that our proposed architecture outperforms prior approaches, or it performs, at least, as efficiently as they do.

Keywords

Supply chain management, Process planning, Scheduling, Transformation matrix, Evolutionary search.

* **Corresponding Author, Email:** m_zandieh@sbu.ac.ir

Introduction

A supply network works via several interrelated entities which are orchestrated as a unified process to turn inputs into predefined demanded outputs from downstream layers of the networks (Beamon, 1998). Considering supply chain, activities involving planning and scheduling are very complicated and must be done through the entire supply chain and also within the enterprise. Thus, in order to achieve global optimal solutions, one needs to focus on more coordinated and integrated planning processes.

Process planning gives insight to how an item is manufactured. It connects design, manufacturing, and scheduling. It also takes alternative resources into account. Generally, customer orders intended for processing might have a flexible process sequence and can be scheduled based on different resources. In Hankins et al. (1984), it is shown that taking advantage of several alternative resources can reduce order–to-delivery times and result in better resource exploitation. Nasr and Elsayed (1990) studied the problem in which each operation could have more than one routing. Their heuristics were meant to deliver an efficient schedule. Brandimarte and Calderini (1995) designed a two-stage tabu search for planning and scheduling. Based on simulated annealing, a method was developed by Palmer (1996) for solving integrated problems. However, in these studies, no rule was employed to sequence an operation. A primal-dual approach was introduced by Guinet (2001) to find the solution for a model of networked production planning. Zhang et al. (2003) purported to present a novel method for integrating these two decisions in a batch production system. Wong et al. (2006) developed an agent-based modeling focusing on negotiations for this integrated problem. Moon et al. (2002) developed a model to sequence and schedule the process in an integrated manner to achieve more flexibility and minimize the total tardiness. Cochran et al. (2003) developed an evolutionary algorithm in which two different populations serve solutions for each stage of bi-objective scheduling problem independently. Tan and Khoshnevis (2004) modeled the integrated problem as an MIP. Moon et al. (2008) designed a genetic algorithm (GA) for PPS to minimize the makespan. Shao et al. (2009) developed a new model to determine PPS decisions jointly, rather than sequentially, and promoted it to IPPS. They presented a genetic algorithm with improved representation to facilitate making these decisions simultaneously.

Leung et al. (2010) proposed the agent-based structure and implemented the ACO algorithm on the platform in which multiple agents operate. They showed that in terms of robustness, an agent-based view of the problem is better than a centralized one. Li et al. (2012) developed an active learning genetic algorithm through learning from excellent individuals/peers to improve the efficiency and quality of the optimization of the IPPS problem. In this study, the cross-over operator was used as the learning operator. Wang et al. (2014) developed a graph based on the ant colony and improved the prematurity of the algorithm through some modifications in attraction process and compared their algorithm with two other types of ACO. Comparing their results with test beds by Leung et al. (2010), they showed improvements in some beds, but not all of them. Zhang et al. (2015) introduced a genetic algorithm in which a chromosome shows the sequence of operation performed on all machines. They showed their GA outperforms the algorithms of other studies in most of the test beds. Zhang and Wong (2016) presented a graph model for IPPS including a network of operation sequences which are evaluated by the mapping rule. The suggested generic framework in their study gives different constructive heuristic methods. Petrovic´ et al. (2016) proposed a particle swarm optimization algorithm equipped with ten different chaotic maps, which was labeled as cPSO to avoid local optima trap. Their multi-objective model is to optimize a blend of five objectives, among which the balancing utilization of machines is less studied in IPPS literature. Xia et al. (2016) developed a hybrid GAVNS to solve the dynamic IPPS problem. They presented a 6-step procedure to deal with IPPS decisions in the more realistic environment with relevant disturbances. While the majority of researchers have studied IPPS in the job shop environment, some have studied other environments i.e., Reconfigurable Manufacturing Systems (RMS) and process industries. In the RMS environment, the multi-configuration nature of machines makes setup planning a critical issue, so that it should be incorporated in process planning. Mohapatra et al. (2013) addressed the adaptive setup planning through cross-machine setup. Bensmaine et al. (2014) developed a heuristic which used availability time and selection index and applied it to a simple simulated instance. There exists a study in process industries done by Shah et al. (2012) that deals with process and scheduling decisions in multi-period mode with a given supply chain network. Here, the purpose of process planning is to assign tasks to sites, and the starting time of processing batches is determined by scheduling. Multi-period modeling used in this study requires authors to consider inventory and transportation decisions. The multi-objective studies on IPPS have been stemmed from the work done by Li et al. (2012). One might classify objectives in such studies into four general classes: makespan and/or flow, machine work/load, tardiness, and the costs of machining or production. To solve these multi-objective problems, researchers have designed GA- and PSO-based algorithms (Li et al. 2012, Mohapatra et al. 2013, Ausaf et al. 2015, Petrovic´ et al. 2016, Luo et al. 2017). In the work done by Ausaf et al. (2015), the authors developed a heuristic algorithm which employed dispatching rules to find satisfactory solutions regarding all classes of objectives, except for the cost. The GA algorithms in the foregoing studies are equipped with some supplement or compliment property, shown in the last column of Table 1. These features help algorithms in conquering the intractability of the problem. Table 1 summarizes the most related IPPS studies in the last five years.

In our work, the model determines three decisions, i.e. the selection of resource as well as the sequencing and scheduling of operation to optimize the total processing time. We also developed an integrated process planning learnable genetic algorithm (IPPLEGA), to tackle the IPPS decisions. IPPLEGA is a development of the learnable genetic architecture (LEGA) presented by Ho et al. (2007), which has been successfully evaluated for solving a number of benchmarks for the Flexible Jobshop scheduling problems (Mesghouni et al. 1997; Chen et al. 1999; Kacem et al. 2002a; Kacem et al. 2002b Brandimarte, 1993) and active learning-based GA presented by Xinyu Li et al (2012). Moreover; in the IPPLEGA we took advantage of a proposed transformation matrix (TM). This evolutionary search is based on TM and able to achieve a satisfactory solution to the model efficiently. The main contribution of the solution method of this study is adapting the application of LEGA in making joint decisions in IPPS by means of introducing innovative module, which is called as “Transformation”. This module manages the relationships of three LEGA modules effectively to solve the IPPS.

The rest of the paper is presented by the following briefly-introduced sections: Section 2 defines the studied problem; section 3 describes an integrated mathematical model dealing with process planning and scheduling; in section 4 the proposed architecture, IPPLEGA, is introduced; section 5 reports computational results, and the conclusion of this study is given in section 6.

Problem definition

The set of processes required for turning all types of inputs into final products can be performed under predefined precedence constraints. Process planning is about identifying operations and sequences based on counterpart routes so that, producing an order is accomplished effectively. IPPS involves determining the best schedules along with process plans with respect to precedence constraints, set of operations, given processing times, and the counterpart routes.

Table 1. IPPS researches during recent five years

Author (Year)	**Environment**	**Other Decisions**	**Objectives**	**Algorithm/ Feature**
Lian et al. (2012)	Job shop		Makespan(S)	ICA/ Operation-based representation
Shah et al. (2012)	Process industry	Production /inventory planning	Total Production/ inventory cost (S)	Lagrangian decomposition / Diagonal Quadratic approximation
Li et al. (2012)	Job shop		Makespan(S)	GA/ active learning
Li et al. (2012)	Job shop		Makespan/MMW/TMW (M)	GA-TS/ Nash Equilibrium solutions for generations
Mohapatra et al. (2013)	RMS	Adaptable setup planning	Machining cost/Makespan/Machine utilization (M)	NSGA-II/ Fuzzy set theory for selecting Pareto solutions
Wang et al. (2014)	Job shop		Makespan(S)	ACO/ graph based
A.Bensmaine et al. (2014)	RMS	time of Availability / selection	Makespan(S)	Heuristic/ Discrete event simulator

Author (Year)	**Environment**	**Other Decisions**	**Objectives**	**Algorithm/ Feature**
		index		
Zhang et al. (2015)	Job shop		Makespan(S)	GA/ object-coding
Ausaf et al. (2015)	Job shop		Makespan/MML/TML/TFT/MFT/TT (M)	Heuristic-PBOA/ Dispatching rules/ External archive
Zhanga and Wong (2016)	Job shop		Makespan(S)	constructive heuristic/ Generic framework
Petrovic´a et al. (2016)	Job shop		Production time, production cost, makespan, machine utilization and mean flow time(M)	cPSO/ Chaotic maps
Xia et al. (2016)	Job shop		Makespan(S)	GAVNS/
Luo et al. (2017)	Job shop		Makespan/TT/TFT/MMW/TMW(M)	MOGA-IE/ Immune principle - External archive

RMS: Reconfigurable Manufacturing Systems
(M): Multi-objective
(S): Single objective

This problem can be described like this: A set of *n* jobs must be processed using *m* machines with alternative operation andcounterpart sequences. It is necessary in a process plan representation to include representation of constraints that limit the decisions to pre-defined precedencies. A pre-determined process sequence governs the manner in which all jobs must be processed. The decision of machine selection is made in the process planning. In our scheme, every job needs a number of operations performable on some pre-defined machines with non-equal processing times. Each machine can perform a number of operations and has finite capacities with different capabilities. The time to complete all customer orders can be usually considered as the summation of process times and some kind of non-value added times including idle and transportation times. When there are several jobs to complete, makespan is the required time to finish all of them.

Model for IPPS

We used an MIP model presented by Moon et al. (2008) which minimizes the makespan. Relations (1) - (8) are adapted from Moon et al. (2008).

$$\text{Minimize } F = \max_{\forall i,k \text{ and } j} \{x_{ikj}\}$$

subject to

$$x_{igq} - x_{ikj} + M(1 - Y_{igq}) \geq p_{igq} \qquad \forall (k,g) \in R_i, i, j \text{ and } q, \qquad (1)$$

$$x_{igq} - x_{ikj} + M(1 - d_{ikg}) \geq p_{igq} \qquad \forall (k,g) \in B_i, i, j \text{ and } q, \qquad (2)$$

$$x_{hgj} - x_{ikj} + \geq p_{hgj} - M(1 - v_{kgj}) \qquad \forall (k,g) \in G_j, i, j, q \text{ and } i \neq h, \qquad (3)$$

$$\textstyle\sum_{j=1}^{H} Y_{ikj} = 1 \qquad \forall i \text{ and } k, \qquad (4)$$

$$x_{ikj} \leq M y_{ikj} \qquad \forall i\,, k, and\ j \qquad (5)$$

$$x_{ikj} \geq \begin{cases} p_{ikj} & \text{for } i, k \in B_i, j \\ 0 & \text{for all other } i, k \end{cases} \qquad (6)$$

$$y_{ikj} \in \{0,1\} \qquad \forall i, k, \text{and } j \qquad (7)$$

$$v_{kgj} \in \{0,1\} \qquad \forall j, k, \text{and } g \qquad (8)$$

For which the following notations stand:

Prime sets:

Jobs: indexed by i, h with the size of I

Operations: indexed by k, g with the size of K and Ki

Machines: indexed by j, q with the size of H

Derived sets:

Precedencies: defined for any job as the pairs of its operations indexed by $\boldsymbol{R_i = \{r_{kg} | \forall g = 1, \dots, J_{1i}\}}$ with the size of J1i

Non-Precedencies: defined for any job as the pairs of its operations, indexed by $\boldsymbol{R_i = \{r_{kg} | \forall g = 1, \dots, J_{1i}\}}$ with the size of $\boldsymbol{J_{2i}}$ which counts the pairs of operations with no precedence relations

Machines - Operations: defined for any machine as the operations which can be processed on it, indexed by Gj

Parameter and variables:

Jobs - Operations - Machines:

The processing time shown by pikj;

The completion time of an operation of each job on each machine calculated by $\boldsymbol{x_{ikj}}$;

A binary variable to determine whether an operation of a job is performed on a machine or not calculated by $\boldsymbol{y_{ikj}}$;

Jobs – Operations - Operations:

Quantifying the precedence requirement of each pair of operations of any job as a binary number by $\boldsymbol{d_{ikg}}$;

Machine - Operations - Operations:

A binary variable to determine the sequence of operations on any machine calculated by $\boldsymbol{v_{kgj}}$;

Constraint (1) forces the model to comply with the given precedence relations for the operations of each job. Whenever $\boldsymbol{Y_{igq}}$ is not equal to 1, the left side of this relation will be sufficiently large and the value of relation is true. Constraint (2) ensures that operations of a job cannot be simultaneously performed, even on different machines. Relation (3) guarantees that a machine can process just one operation of any job. Constraint (4) results in assigning just one machine to each operation. Relation (5) sets the completion time of each operation on each machine to zero if no such an assignment is adjusted. Constraints (6) – (8) set problem variables as integer or binary ones.

Integrating learning population evolution and process planning

Integrating well-devised knowledge-based encoding, learning methodology and initialization results in improving the quality of the algorithm solution and its efficiency (Ho et al. 2007). In order to solve flexible job-shop problem (FJSP), Ho et al. (2007) proposed an architecture called LEGA. Three modules constitute LEGA, namely a Population Generator module, Evolutionary Algorithm (EA) module and a Schemata Learning (SL) module. These modules require a feasible operation sequence in order to solve a problem. To do this, we added a new module, i.e. Transformer module, to the structure of LEGA (see Figure 1). Transformer module provides an appropriate sequence for these modules. Sections 4.1, 4.2, 4.3, and 4.4 provide information on the modules mentioned above.

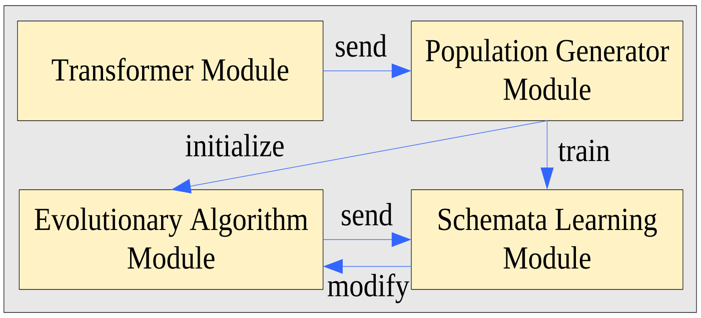


Figure 1. IPPLEGA architecture

Module 1: Population Generator

A heuristic algorithm known as composite dispatching rules (CDR) is employed to produce proper initial solutions to elaborate the scheduling suitable flexible job-shop problem. CDR was proposed by Ho et al. (2007) and its effectiveness was evaluated on several benchmarks proposed by Mesghouni et al. (1997) and Kacem et al. (2002b).

With respect to CDR, note that three basic dispatching rules, namely SPT, LPT and FIFO, are applied. Thus, with one random set of n jobs to be scheduled, we will be able to generate three different schedules.

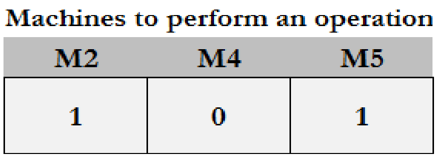
Module 2: Schemata learning

Two distinguished memories, i.e. chromosomal memory and operational memory, were used by Ho et al. (2007) and yielded solutions with high quality for different instances of FJSP. Therefore, we took these two memories into consideration. Additionally, obtained results show that the use of chromosomal and operational memories in our proposed architecture can lead to better solutions.

The main purpose of incorporating chromosomal memory in our work is to ensure the survival of high quality chromosomes into the next generation. In order to achieve this goal, a similarity template is operationalized during the crossover process. A predefined number of solutions with better fitness function values are selected in the current generation so that the similarity template of these chromosomes are inherited and the current population is influenced. Each chosen chromosome is evaluated based on its desirability against other chromosomes in chromosomal memory. The first k (a predefined value) chromosomes in term of similarities will be moved to the current generation. We use the following equation to calculate the similarity in term of Hamming distance:

$$\boldsymbol{S}\,(\boldsymbol{C1},\boldsymbol{C2}) = \sum_{i=1}^{n} \boldsymbol{Opt1}(\boldsymbol{i}) \otimes \boldsymbol{Opt2}(\boldsymbol{i}) + \sum_{j=1}^{m} \boldsymbol{Mch1}(\boldsymbol{j}) \otimes \boldsymbol{Mch2}(\boldsymbol{j})$$

where C1 and C2 are two chromosomes. Each chromosome has two main parts, i.e. operation order and machine order. By positioning each operator and machine, the chromosome gives a unique solution. By comparing the positions one by one and finding the same operation in the operation part and the machine in the machine part, the function returns 1, otherwise the returned value is 0. The smaller value of S indicates the closer chromosomes. Operational memory includes a set of machines which are technically capable to process a particular operation of a job. The structure of operation memory is an array of bites. As a matter of fact, this memory keeps track of more proper machines for processing the operations. In order to add any other machine to the operational memory, Ho et al. (2007) considered the processing time; but in this work, we took the end-time into consideration. As an example, assume that the set of machines to perform one particular operation of job i is $\{\boldsymbol{M_2}, \boldsymbol{M_4}, \boldsymbol{M_5}\}$. Moreover, assume that in a generation, this operation is processed on M2, M5. Figure 2 shows the operational memory for this operation. Furthermore, to reduce the computational time, memories are updated every q (a predefined value) generation.



Machines to perform an operation

M2	M4	M5
1	0	1

Figure 2. Operation memory for a particular operation

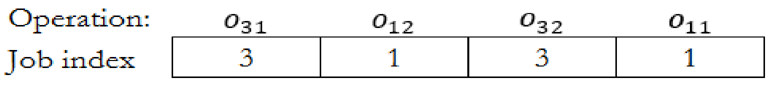
Module 3: Evolutionary algorithm (EA)

The third module of IPPLEGA architecture contains a genetic algorithm. In order to have high-quality solutions, we also used active scheduling in this module. The details of GA are illustrated below.

Encoding

Previous chromosomal representations for solving the FJSP include a parallel machine and a parallel job representation by Mesghouni et al. (1997), A–B string representation proposed by Chen et al. (1999), and an assignment table introduced by Kacem et al. (2002b). An important drawback of these representations is that after each crossover and mutation, we will have infeasible solutions most of the time.

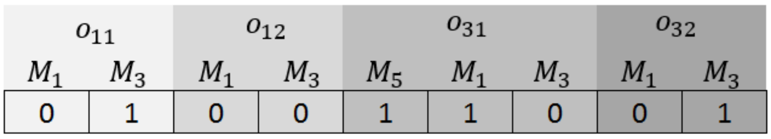
The chromosomal representation designed by Ho et al. (2007), i.e. OOMS, guarantees feasible chromosomes in the randomized crossover and mutation procedures. Moreover, this representation helps to construct active schedules while running the decoding algorithm. OOMS consists of two components, namely the operation order and the machine selection. In this paper, the representation of operation order is according to Cheng et al. (1996), Bierwirth (1995), and Varela et al. (2003). In Ho et al. (2007) an individual is generated by substituting each operation with its job index. As an example, consider four operations O11, O12, O31, and O32, and let $\{\boldsymbol{O_{31}}, \boldsymbol{O_{12}}, \boldsymbol{O_{32}}, \boldsymbol{O_{11}}\}$ be a possible schedule. The resulting process sequence will be $\{\mathbf{3}, \mathbf{1}, \mathbf{3}, \mathbf{1}\}$. Figure 3 illustrates this example.



Operation:	o_{31}	o_{12}	o_{32}	o_{11}
Job index	3	1	3	1

Figure 3. An operation part of a chromosome

The machine selection part of a chromosome is built up by binary values. For instance, Let O11, O12, O31, and O32 be our operations. M1, M3, and M5 are three machines meant to perform the operations. Suppose that O11, O31, and O32 can be processed on M1 and M3, and O12 can be processed on all machines. If O11 is processed on M3, O12 is processed on M5, M1 performs the O31, and O32 is assigned to M3. Machine selection part is shown in Figure 4.



o_{11}		o_{12}		o_{31}			o_{32}	
M_1	M_3	M_1	M_3	M_5	M_1	M_3	M_1	M_3
0	1	0	0	1	1	0	0	1

Figure 4. A machine selection part of a chromosome

Crossover operator

To encode a solution, the applied chromosome consists of two parts. Therefore, in this study, to apply crossover to operation order part, we used a two-point crossover, in line with Varela et al. (2003),. For instance, assume that we have these two chromosomes: (2 1 2 1 1) and (1 1 1 2 2). Supposed we have (2 1 1) as a randomly selected substring from (2 1 2 1 1). So, O22, O12, and O13 are realized from the substring. Then, after omitting the corresponding positions of the elements in the second chromosome, we have (1 1 1 2 2). Finally, a new offspring is generated by inserting the substring to the second chromosome at the same position in the first chromosome: (1 2 2 1 1). Machine selection part of the chromosome is crossed-over by generating two random numbers between 2 and the length of machine selection part minus one. Two partial parts of the chromosomes between these two loci are then exchanged. Figure 5 depicts an example in which r1 = 2 and r2 = 4.

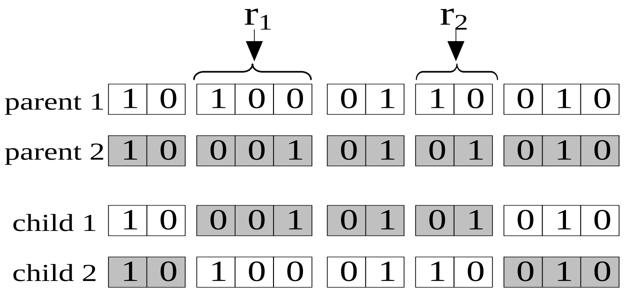

r1
r2
parent 1 1 0 1 0 0 0 1 1 0 0 1 0
parent 2 1 0 0 0 1 0 1 0 1 0 1 0
child 1 1 0 0 0 1 0 1 0 1 0 1 0
child 2 1 0 1 0 0 0 1 1 0 0 1 0

Figure 5. An example of mutation operator on machine selection part

Mutation operator

Mutation explores the solution space via being incorporated on operation and machine parts of a chromosome. Operation order part is mutated, by inverting a substring of a chromosome. This substring is determined by generating two random numbers between 2 and the length of operation order part minus one. For instance, if we consider (3 1 2 2 1) and assume 1 and 3 as the generated random positions, the chromosome will be mutated as (2 1 3 2 1).

The operational memory is activated in the mutation process for the machine selection part as illustrated by the flow chart shown in Figure 6. The applied notation in Figure 6 is as follows:

Mk: the machine that processes Oik in the given chromosome;

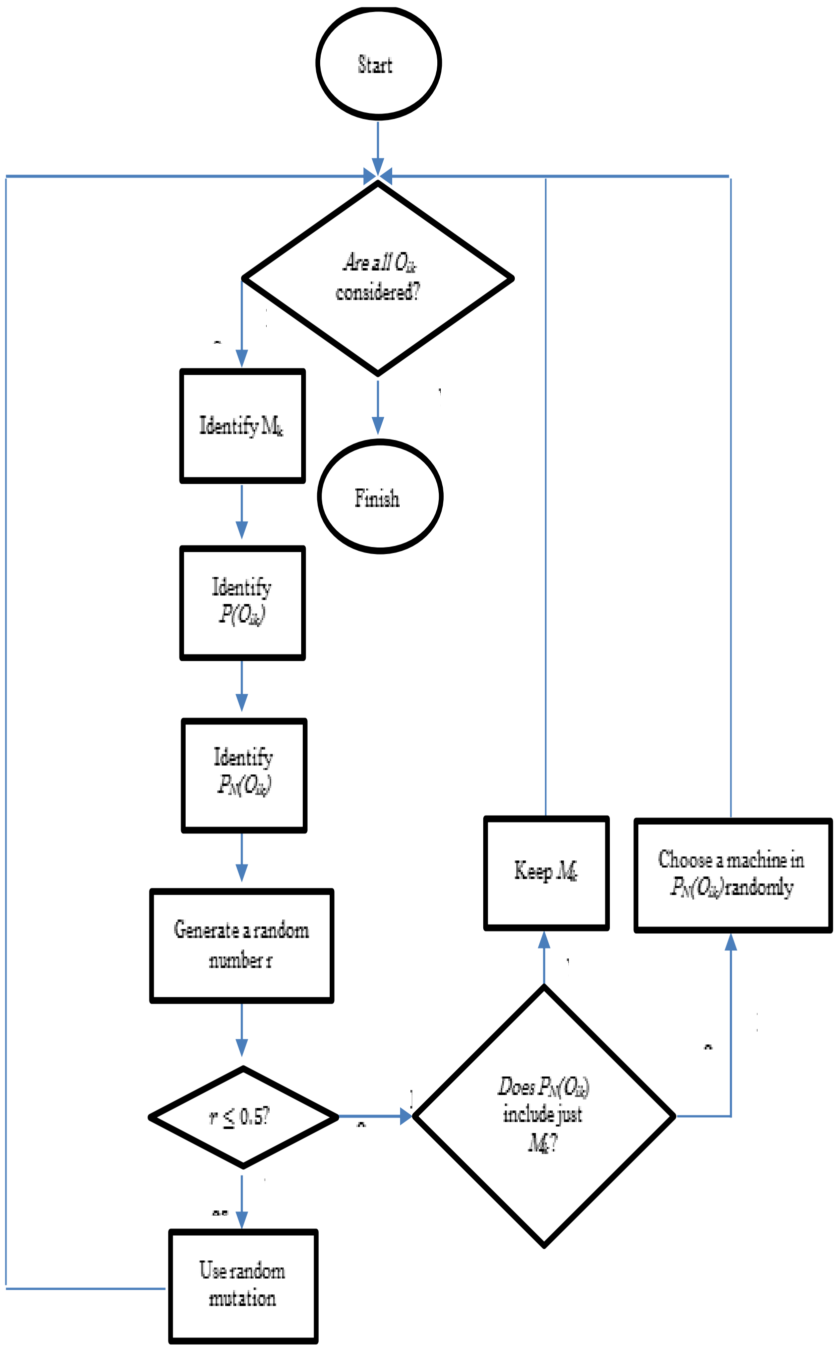
P (Oik): the set of machines that can be selected to perform Oik;

PN (Oik): the set of machines that are more capable according to operational memory for performing Oik.

According to this figure, the chance of changing the machine assigned to perform Oik is increased. It would be changed with the probability of 0.5 during the random mutation or with the same probability through the influence of operational memory.

Active scheduling

Ho et al. (2007) made use of active schedule. Active solutions include simultaneously zero-tardiness and optimal solutions. Thus, they prove to be extremely responsive. Figure 7 presents the flowchart for decoding an OOMS to obtain an active FJSP schedule. According to Ho et al. (2007), first a time gap is computed to complete the schedule with inserting a new operation. In case that insertion is not possible, the operation is added to the end of the currently scheduled operations.



Start
Are all O_{ik} considered?
Identify M_k
Finish
Identify $P(O_{ik})$
Identify $P_N(O_{ik})$
Generate a random number r
$r \leq 0.5$?
Use random mutation
Does $P_N(O_{ik})$ include just M_k?
Keep M_k
Choose a machine in $P_N(O_{ik})$ randomly

Figure 6. Activating the operational memory in mutation process

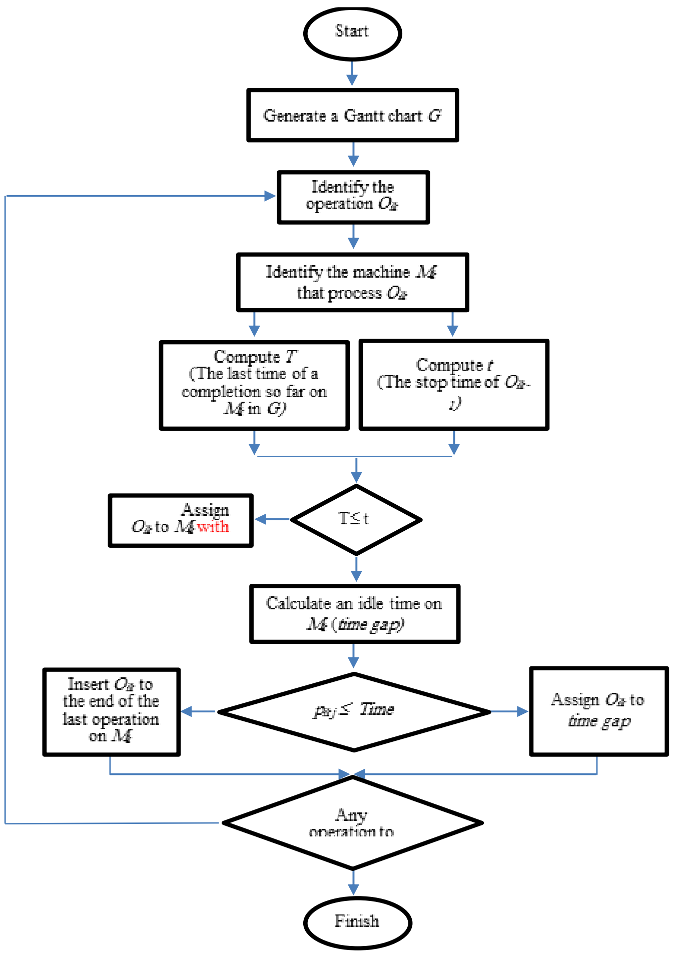

Start
Generate a Gantt chart G
Identify the operation O_{ik}
Identify the machine M_j that process O_{ik}
Compute T (The last time of a completion so far on M_j in G)
Compute t (The stop time of O_{ik-1})
T≤ t
Assign O_{ik} to M_j with
Calculate an idle time on M_j (*time gap*)
$p_{ikj} \leq$ *Time*
Insert O_{ik} to the end of the last operation on M_j
Assign O_{ik} to *time gap*
Any operation to
Finish

Figure 7. Decoding an OOMS to obtain an active FJSP schedule (Ho et al. 2007)

Transformer module

As shown in figure 1, this module adapts a process planning problem to be solved by the modules population generator, EA, and SL. The relations among modules in this figure without considering the transformer module have been applied in pure job shop scheduling context by researchers in the existing literature. In fact, this module generates a feasible sequence for each job with respect to its operation network as an input to population generator module. We propose a procedure, i.e. Transformation, for this adaptation. Performance of this procedure revolves around a matrix called transformation matrix (TM) formed prior to the performance of the Transformation procedure.

The size of TM is (K+1)(k) or $(number\ of\ operations + 1) \times (number\ of\ operations)$. Each column of the matrix is related to one of the operations. The first row of this matrix is named *blank* and it means the cells of this row are initially equal to 0. Each of the next rows is related to one operation. The following rule is applied to determine the value of an entry in one of these rows:

The entry in the (k+1)th row and gth column is determined as $TM_{k+1,g}$. This cell is equal to 1 if operation k precedes operation , otherwise it is set to 0.

We use transformation procedure to generate feasible operation sequences for the jobs. This procedure receives TM as an input. The output of the procedure is a feasible operation sequence for the job. Transformation procedure is described in Figure 8.

```
Procedure: Transformation
  Input data
       Transformation Matrix;
       Let $n_i$ be the number of operations in job $i$;
         For $k$=1 to $n_i$ do
             Begin
             If there are no columns in TM with all
               entries equal to 0, it means this is
               infeasible, then stop;

             Else select a column in TM whose all
               entries are equal to 0
                 (if there are 2 or more columns with
                 this property, pick one at random);
                 Set the corresponding entry in the
                 blank to 1;
                 If there is an entry in the
                 corresponding row of TM with its
                 value set to 1, change its value to 0;
                 The operation which corresponds to
                 the selected column is the output;

             End
         End
```

Figure 8. Transformation procedure

As an example and for more detailed explanation, consider the operation network of an arbitrary job, shown in Figure 9, which consists of four operations .

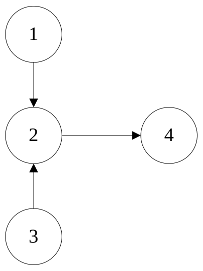


Figure 9. The operation network of an arbitrary job

Based on Figure 9 and the rule given above, Transformation Matrix will be

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Entries in the first row of this matrix, $blank$, are set to 0. Columns 1, 2, 3, and 4 correspond to operations 1, 2, 3, and 4 respectively. Moreover, rows 2, 3, 4, and 5 correspond to operations 1, 2, 3, and 4, respectively. Now, Transformation procedure is applied to this matrix. All the entries in columns 1 and 3 are 0. Suppose column 3 (operation 3) is picked. TM1,3 is, then, set to 1. Then the 4th row (which corresponds to operation 3) will be considered; since $TM_{4,2} = 1$, it will be set to 0. The output of this procedure will be operation 3 and Transformation Matrix will be

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The same way is followed until all operations are considered. There will be two possible outcomes when Transformation procedure is applied to this Transformation Matrix: $1 \rightarrow 3 \rightarrow 2 \rightarrow 4$ or $3 \rightarrow 1 \rightarrow 2 \rightarrow 4$ which are both feasible operation sequences according to Figure 9.

Experiments and results

To show the quality of our algorithm, all the problems that were solved by Moon et al. (2008) and Shao et al. (2009) are tested. We, then, compare the results and show that our proposed approach is superior to previous methods or is at least as efficient as they are.

Our proposed approach was coded in C#.net on a 2.00 GHz Intel Dual CPU desktop computer with a 1.00 GB RAM.

The computation times for all experiments were very little such that in the scale of one second, we cannot distinguish between them.

Experiment 1

A problem with five jobs and five machines was used for the first experiment. We named this problem Prob1, 1. To complete each job, operations with different predefined orders are required. Let Oik ($1 \leq i \leq 5, 1 \leq k \leq 4$) be the kth operation for the ith job. Figure 10 shows the operation precedencies for jobs. The operational routings of each job and machine options are shown in Table 2.

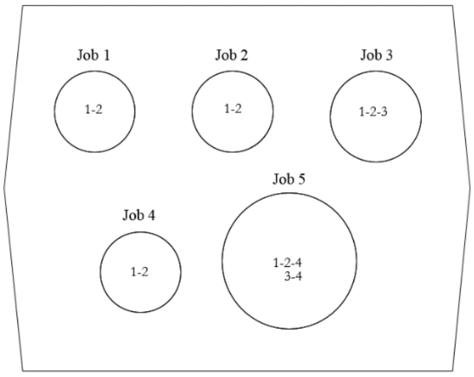


Figure 10. Operation precedencies for jobs

Table 2. The operational routing of each job

Job	Operations	Machine options for operations (Operation, Machine, Time)	Sequence options
1	1,2	(*1*, 1, 5), (*1*, 2, 3) (*2*, 2, 5)	*1-2*
2	1,2	(*1*, 3, 6), (*1*, 4, 5) (*2*, 5, 4)	*1-2*
3	1,2,3	(*1*, 2, 4), (*1*, 1, 5) (*2*, 3, 2), (*2*, 4, 3) (*3*, 2, 5)	*1-2-3* *1-3-2* *3-1-2*
4	1,2	(*1*, 3, 4) (*2*, 4, 5)	*1-2*
5	1,2,3,4	(*1*, 1, 4), (*1*, 2, 3) (*2*, 1, 2), (*2*, 2, 4) (*3*, 3, 5) (*4*, 3, 4), (*4*, 5, 3)	*1-2-3-4* *1-3-2-4* *3-1-2-4*

Best makespan obtained by Moon et al. (2008) for this problem was 16; but our proposed approach yielded a better makespan, i.e. 14. Figure 11 shows the schedule output by Gantt chart for this problem.

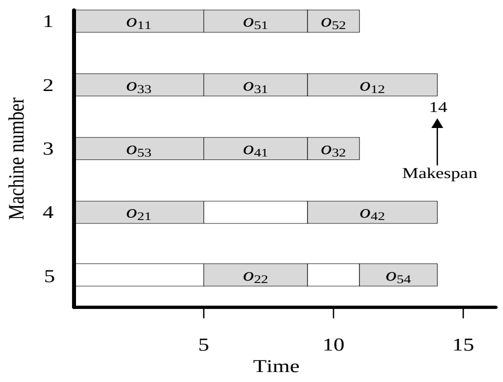


Figure 11. Gantt chart for experiment 1

Our result and Figure 11 show that the best sequence of operations is selected to perform each job and our proposed architecture significantly improves the makespan of the schedule.

Experiment 2

Like Moon et al. (2008), in Table 3 several samples were implemented by adjusting genetic parameters (Popsize, Maxgen, Pc and Pm) to various values. Pc and Pm are the probability of cross-over and mutation.

Table 3. Experimental results under various job sizes

Proble m name	1	2	3	4	5	p_c	p_m	6
Prob$_{2,1}$	8	20	5	50 50 10 0 10 0 15 0	100 0 100 0 150 0 150 0 200 0	0. 4 0. 5 0. 4 0. 5 0. 4	0. 1 0. 5 0. 1 0. 5 0. 1	27 27 27 27 27
Prob$_{2,2}$	1 6	40	5	50 50 10 0 10	100 0 100 0 150	0. 4 0. 5 0.	0. 1 0. 5 0.	50 50 50 50 50

				0	0	4	1	
				150	1500	0.5	0.5	
					2000	0.4	0.1	
Prob$_{2,3}$	32	80	5	50	1000	0.4	0.1	100
				50	1500	0.4	0.1	100
				100	1500	0.4	0.1	100
				150	2000	0.4	0.1	100
				150	3000	0.4	0.1	100
Prob$_{2,4}$	64	160	5	50	1000	0.4	0.1	203
				50	1500	0.4	0.1	203
				100	1500	0.4	0.1	203
				150	2000	0.4	0.1	203
				150	3000	0.4	0.1	203

1: Job; 2: Operation; 3: Machine; 4: pop_size; 5: max_gen; 6: Makespan

In Table 2, the first problem had eight jobs, twenty operations, and five machines. Just like the previous experiment, let Oik ($1 \leq i \leq 8$, $1 \leq k \leq 4$) be the kth operation for the ith job. Figure 12 shows the operation precedencies for jobs. The operational routings of each job and machine options are shown in Table 4.

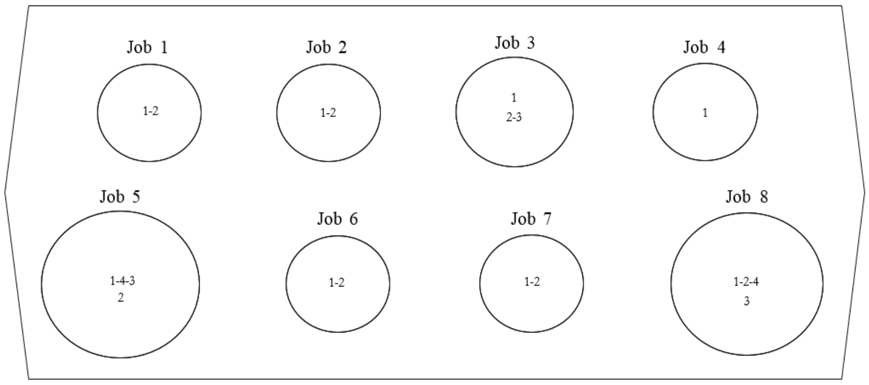


Figure 12. Operation precedencies for jobs

Table 4. The operational routing of each job

Job	Operations	Machine options for operations (Operation, Machine, Time)	Sequence options
1	*1*, *2*	(*1*, 1, 5), (*1*, 2, 3) (*2*, 2, 7)	*1-2*
2	*1*, *2*	(*1*, 3, 6) (*2*, 4, 3), (*2*, 5, 4)	*1-2*
3	*1*, *2*,*3*	(*1*, 1, 7) (*2*, 2, 4), (*2*, 3, 6) (*3*, 3, 7), (*3*, 4, 7)	*1-2-3* *2-1-3* *2-3-1*
4	1	(*4*, 5, 10)	*1*
5	*1*, *2*,*3*,*4*	(*1*, 1, 4), (*1*, 2, 5), (*1*, 3, 8) (*2*, 4, 5) (*3*, 4, 6), (*3*, 5, 5) (*4*, 5, 4)	*1-2-3-4* *1-3-2-4* *2-1-3-4* *2-3-1-4* *3-2-1-4* *3-1-2-4*
6	*1*, *2*	(*1*, 2, 2), (*1*, 3,6) (*2*, 3, 8)	*1-2*
7	*1*, *2*	(*1*, 3, 3), (*1*, 4, 8) (*2*, 4, 7), (*2*, 5, 4)	*1-2*
8	(1, 2, 3, 4)	(1, 1, 3), (1, 3, 5) (2, 3, 7) (3, 4, 9), (3, 5, 6) (4, 5, 3)	1-2-3-4 1-3-2-4 3-1-2-4

When applied to this problem, our approach resulted in 27 as the value of the minimized makespan. Figure 13 depicts the schedule output by Gantt chart for this problem.

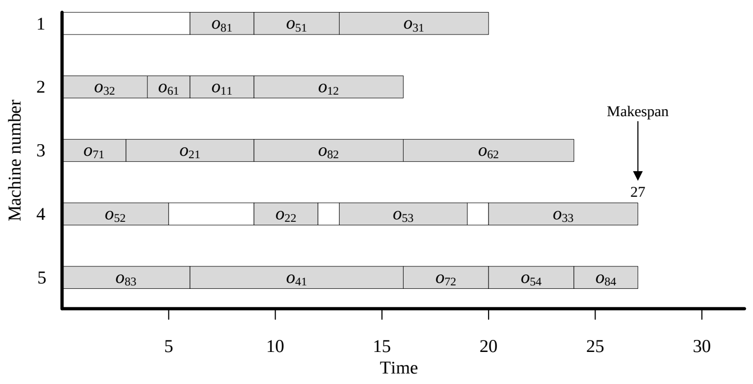


Figure 13. A Schedule on 5 machines for the first problem in Table 2

Based on the reasoning given in Moon et al. (2008) and with respect to Table 2, since this result had no variation considering various settings of genetic parameters, our approach generates the best makespan in a robust manner. This result is supported by other computations in various environments. Moreover, we realized that the presented architecture had an effective performance while generating the best makespan under different conditions.

Experiment 3

For the third experiment, we applied our approach to a problem, Prob3, 1, which has been previously taken into account in Moon et al. (2008). This problem consisted of five jobs and five machines. Also, the operational routing of each job and the alternative sequences are shown in Table 5.

Our approach could obtain 33 for makespan, which is the same value that was obtained by Moon et al. (2008). This result is illustrated in Figure 14.

Table 5. The operational routing of each job (Moon et al, 2008)

Job	Operations	Machine options for operations (Operation, Machine, Time)	Sequence options
1	1,2,3,4	(*1*, 1, 5), (*1*, 2, 3) (*2*, 2, 7) (*3*, 3, 6) (*4*, 4, 3), (*4*, 5, 4)	*1-2-2-3*
2	1,2,3,4	(*1*, 1, 7) (*2*, 2, 4), (*2*, 3, 6)	*1-2-2-3*

		(*3*, 3, 7), (*3*, 4, 7) (*4*, 5, 10)	
3	1,2,3,4	(*1*, 1, 4), (*1*, 2, 5), (*1*, 3, 8) (*2*, 4, 5) (*3*, 4, 6), (*3*, 5, 5) (*4*, 5, 4)	*1-2-2-3*
4	1,2,3,4	(*1*, 2, 2), (*1*, 3, 6) (*2*, 3, 8) (*3*, 3, 3), (*3*, 4, 8) (*4*, 4, 7), (*4*, 5, 4)	*1-2-2-3*
5	1,2,3,4	(*1*, 1, 3), (*1*, 3, 5) (*2*, 3, 7) (*3*, 4, 9), (*3*, 5, 6) (*4*, 5, 3)	*1-2-2-3*

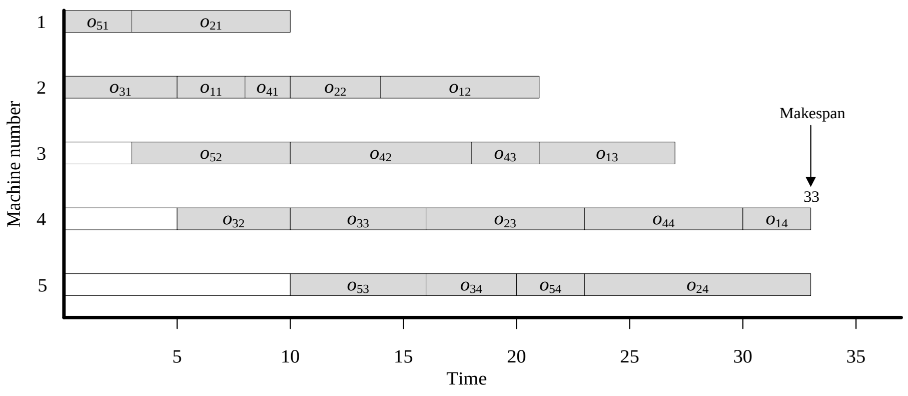


Figure 14. A Schedule on 5 machines for Prob3, 1 in Moon et al. (2008)

Experiment 4

The problem Prob4, 1 in Shao et al. (2009) considers eight machines with six jobs which have flexible routings. The approach presented in Shao et al. (2009) yielded a value of 162 as the makespan, and our proposed architecture resulted in the same value.

Experiment 5

The problem, Prob5, 1, in Shao et al. (2009) considers six machines with five jobs requiring 21 operations to be completed. Our approach and the one given in Shao et al. (2009) both obtained 28 as the value of the minimized makespan.

Experiment 6

This problem is also taken from Shao et al. (2009). A problem with four jobs and three machines, Prob6, 1, was used for this experiment. Best makespan obtained by our proposed architecture was 1100 which is equal to the value given in Shao et al. (2009).

Comparative results between the approaches

We applied our proposed architecture to all the problems mentioned in Moon et al. (2008) and Shao et al. (2009). Gained results imply that our proposed approach is efficient and also superior to the approach given in Moon et al. (2008). According to the obtained results, it can be concluded that the performance of our architecture is as efficient as the presented approach in Shao et al. (2009). Table 6 and Table 7 give the comparative results between the approaches along with the percentage improvements in the results. A percentage improvement is given by

$$\frac{(Value\ obtained\ by\ prior\ approach\ )-(Value\ obtained\ using\ IPPLEGA)}{(Value\ obtained\ by\ prior\ approach\ )} \times 100$$

Table 6. Comparative results between IPPLEGA and the proposed approach in Moon et al. (2008)

	Solution (Makespan)		**% Improvements**
Problem Name	IPPLEGA	Moon et al. (2008)	
Prob$_{1,1}$	14	16	12.5%
Prob$_{2,1}$	27	34	20.59%
Prob$_{2,2}$	50	63	20.63%
Prob$_{2,3}$	100	114	12.28%
Prob$_{2,4}$	203	227	10.57%
Prob$_{3,1}$	33	33	0%

Table 7. Comparative results between IPPLEGA and the proposed approach in Shao et al. (2009)

	Solution (Makespan)		**% Improvements**
Problem Name	IPPLEGA	Shao et al. (2009)	
Prob$_{4,1}$	162	162	0%
Prob$_{5,1}$	28	28	0%
Prob$_{6,1}$	1100	1100	0%

Conclusions and future works

This paper presented an architecture named IPPLEGA. The goal we aimed to reach by developing IPPLEGA was to obtain the optimal machine assignments and operation sequences and to find a schedule so that the makespan is minimized. IPPLEGA consists of some memories and a class of scheduling methods, i.e. active scheduling. This architecture helped us to improve results obtained by GA. Our devised approach led us to the results that successfully dominate the solutions given by previous methods or, at least, are as good as the results obtained by prior approaches. The experimental results have also pointed out that combining memories and active scheduling can be an efficient method while aiming to solve IPPS through GA.

An interesting research topic would include other evolutionary algorithms. Another direction for the future research would be considering other objective functions. It is interesting in the world of standardization to investigate the efficiency of the variants of design schemes with their corresponding routings integrated with scheduling decisions.

References

Ausaf, M. F., Gao, L., & Li, X. (2015). Optimization of multi-objective integrated process planning and scheduling problem using a priority based optimization algorithm. *Frontiers of Mechanical Engineering, 10*(4), 392–404.

Beamon, B. M. (1998). Supply chain design and analysis: Models and methods. *International Journal of Production Economics 55*, 281–294.

Bensmaine, A., Dahane, M., & Benyoucef, L. (2014). A new heuristic for integrated process planning and scheduling in reconfigurable manufacturing systems. *International Journal of Production Research, 52*(12), 3583-3594.

Bierwirth, C. (1995). A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spektrum, 17*,87-92.

Brandimarte, P., (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research, 22*, 158-183.

Brandimarte, P., & Calderini, M. (1995). A heuristic bi-criterion approach to integrated process plan selection and job shop scheduling. *International Journal of Production Research 33*, 161–181.

Chen, H., Ihlow, J., & Lehmann, C. (1999). A genetic algorithm for flexible job-shop scheduling. *Proceedings of the IEEE International Conference on Robotics and Automation 2*, 1120-1125.

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job shop scheduling problems using genetic algorithms. I. representation. *Computers and Industrial Engineering, 30*(4), 983-997.

Cochran, J. K., Horng, S., & Fowler, J. W. (2003). A multi-population GA to solve multi-objective scheduling problems for parallel resources. *Computers and Operations Research 30*, 1087–1102.

Guinet, A. (2001). Multi-site planning: A transshipment problem. *International Journal of Production Economics, 74*,21–32.

Hankins, S. L., Wysk, R. A., & Fox, K. R. (1984). Using a CATS database for alternative machine loading. *Journal of Manufacturing Systems, 3*, 115–120.

Ho, N. B., Tay, J. C., & Lai, E. M. (2007). An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research, 179*, 316–333.

Kacem, I., Hammadi, S., & Borne, P. (2002a). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man and Cybernetics, 32*(1), 1-13.

Kacem, I., Hammadi, S., & Borne, P. (2002b). Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, *60*, 245-276.

Mesghouni, k., Hammadi, S., & Borne, P. (1997). Evolution programs for job-shop scheduling. *Proceedings of the IEEE International Conference on Computational Cybernetics and Simulation, 1*, 720–725.

Leung, C. W., Wong, T. N., Mak, K. L., & Fung, R. Y. K. (2010). Integrated process planning and scheduling by an agent-based ant colony optimization. *Computers and Industrial Engineering, 59*(1), 166-180.

Li, X., Gao, L., & Shao, X. (2012). An active learning genetic algorithm for integrated process planning and scheduling, *Expert Systems with Applications, 39*(8), 6683-6691.

Lian, K., Zhang, C., Gao, L., & Li, X. (2012). Integrated process planning and scheduling using an imperialist competitive algorithm. *International Journal of Production Research, 50*(15), 4326-4343.

Luo, G., Wen, X., Li, H., Ming, W., & Xie, G. (2017). An effective multi-objective genetic algorithm based on immune principle and external archive for multi-objective integrated process planning and scheduling. *The International Journal of Advanced Manufacturing Technology, 91*(9–12), 3145–3158.

Mohapatra, P., Benyoucef. L., & Tiwari, M.K. (2013). Integration of process planning and scheduling through adaptive setup planning: A multi-objective approach. *International Journal of Production Research, 51*, 23-24.

Moon, C., Kim, J., & Hur, S. (2002). Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain. *Computers and Industrial Engineering, 43*,331–349.

Moon, C., Lee, Y. H., Jeong, C. S., & Yun, Y. (2008). Integrated process planning and scheduling in a supply chain. *Computers and Industrial Engineering, 54*, 1048-1061.

Nasr, N., Elsayed, A. (1990). Job shop scheduling with alternative machines. *International Journal of Production Research, 28*, 1595–1609.

Palmer, G. J. (1996). A simulated annealing approach to integrated production scheduling. *Journal of Intelligent Manufacturing, 7*, 163–176.

Petrović, M., Vuković, N., Mitić, M., & Miljković, Z. (2016). Integration of process planning and scheduling using chaotic particle swarm optimization algorithm. *Expert Systems with Applications, 64*, 569-588.

Shah, N. K., & Ierapetritou, M. G. (2012). Integrated production planning and scheduling optimization of multisite, multiproduct process industry. *Computers and Chemical Engineering, 37*, 214-226.

Shao, X., Li, X., Gao, L., & Zhang, C. (2009). Integration of process planning and scheduling-A modified genetic algorithm-based approach. *Computers and Operations Research, 36*, 2082-2096.

Tan, W., & Khoshnevis, B. (2004). A linearized polynomial mixed integer programming model for the integration of process planning and scheduling. *Journal of Intelligent Manufacturing*, *15*, 593–605.

Varela, R., Vela, C. R., Puente, J., & Goméz, A. (2003). A knowledge based evolutionary strategy for scheduling problems with bottlenecks. *European Journal of Operational Research*, *145*(1), 57-71.

Wang, J., Fan, X., Zhang, C., & Wan, S. (2014). A Graph-based Ant Colony Optimization Approach for Integrated Process Planning and Scheduling. *Chinese Journal of Chemical Engineering*, *22*(7), 748-753.

Wong, T. N., Leung, C. W., Mak, K. L., & Fung, R. Y. K. (2006). An agent-based negotiation approach to integrate process planning and scheduling. *International Journal of Production Research*, *44*(7), 1331-1351.

Xia, H., Li, X., & Gao, L. (2016). A hybrid genetic algorithm with variable neighborhood search for dynamic integrated process planning and scheduling, *Computers and Industrial Engineering*, *102*, 99-112.

Zhang, L., & Wong, T. N. (2015). An object-coding genetic algorithm for integrated process planning and scheduling. *European Journal of Operational Research*, *244*(2), 434-444.

Zhang, L., & Wong, T. N. (2016). Solving integrated process planning and scheduling problem with constructive meta-heuristics. *Information Sciences 340–341*, 1-16.

Zhang, Y. F., Saravanan, A. N., & Fuh, J. Y. H. (2003). Integration of process planning and scheduling by exploring the flexibility of process planning. *International Journal of Production Research*, *41*(3), 611-628.