AN ENHANCED GENETIC ALGORITHM WITH AN INNOVATIVE ENCODING STRATEGY FOR FLEXIBLE JOB-SHOP SCHEDULING WITH OPERATION AND PROCESSING FLEXIBILITY

XUEWEN HUANG AND XIAOTONG ZHANG
Faculty of Management and Economics, Dalian University of Technology
Dalian, Liaoning 116023, China
SARDAR M. N. ISLAM*
ISILC, Victoria University
Melbourne, Australia
CARLOS A. VEGA-MEJÍA
ISILC, Victoria University
Melbourne, Australia
Operations & Supply Chain Management Research Group
Universidad de La Sabana
Bogotá, Colombia

(Communicated by Frank Werner)

Abstract. This paper considers the Flexible Job-shop Scheduling Problem with Operation and Processing flexibility (FJSP-OP) with the objective of minimizing the makespan. A Genetic Algorithm based approach is presented to solve the FJSP-OP. For the performance improvement, a new and concise Four-Tuple Scheme (FTS) is proposed for modeling a job with operation and processing flexibility. Then, with the FTS, an enhanced Genetic Algorithm employing a more efficient encoding strategy is developed. The use of this encoding strategy ensures that the classic genetic operators can be adopted to the utmost extent without generating infeasible offspring. Experiments have validated the proposed approach, and the results have shown the effectiveness and high performance of the proposed approach.

1. Introduction. Process planning and scheduling are two key functions both responsible for the efficient allocation and utilization of manufacturing resources. Process planning determines how a part is to be manufactured in accordance with its design specifications, while scheduling assigns manufacturing resources and schedules operations so that some relevant criteria are satisfied, such as minimum makespan and minimum mean flow time. Integration of these two functions can significantly enhance the performance of a flexible job-shop manufacturing system in terms of

2020 Mathematics Subject Classification. Primary: 90B35, 90C59; Secondary: 68M20.
Key words and phrases. IPPS, flexible job-shop scheduling, operation flexibility, processing flexibility, Genetic Algorithm.

Xuewen Huang is supported by the National Science and Technology Plan of China under Grant No. 2015BAF09B01 and the China Scholarship under Grant No. 201606060170.
* Corresponding author: Sardar M. N. Islam.
reduction of production cost, bottleneck elimination, balanced level of machine utilization, and facility productivity [14, 57, 35, 33]. In the last few decades, more attention has been paid to the problem of the Integrated Process Planning and Scheduling (IPPS). The IPPS problem mainly stems from three types of flexibility: operation flexibility, sequencing flexibility and processing flexibility [6]. Operation flexibility refers to the possibility of performing an operation on different machines with identical or non-identical processing times. Sequencing flexibility corresponds to the possibility of interchanging the sequence in which the required manufacturing operations are performed. Processing flexibility refers to the possibility of producing the same manufacturing feature with alternative operations or sequences of operations.

IPPS aims to optimize process planning and scheduling simultaneously. This causes the IPPS problem to be strongly NP-hard and even more difficult to find an optimal solution [29, 28]. Generally, there are three main types of solution approaches for the problem [56, 66]: simulation-based approach [29, 17], meta-heuristic-based approach [31, 58, 40, 39, 55, 71, 72, 59, 44] and agent-based approach [63, 64, 41, 45]. Among these approaches, the meta-heuristic approach has been recognized as the most important mechanism for the IPPS problem. A typical and widely-used type of meta-heuristic is Evolutionary Computation (EC), such as Genetic Algorithms (GAs), evolution strategies, evolutionary programming and co-evolutionary algorithms. EC algorithms are inspired by Darwin’s evolutionary theory, where a chromosome (or individual) is used to map the target problem and generate a solution by applying an encoding/decoding strategy, and a population of individuals is modified through genetic operators, such as recombination (crossover) and mutation. ECs have been successfully adopted to solve the IPPS problem, as proven by the growing number of papers on the topic. Examples include GAs [48, 35, 32, 34, 58, 55, 40, 39, 71, 72, 60, 10, 67], Symbolic Evolutionary Algorithms (SEAs) [31, 30], and hybrid approaches of EC [37, 1, 49]. However, the current encoding strategies for the IPPS problem reported in the literature make almost all the classic genetic operators inapplicable due to the fact that a vast of offspring will be infeasible or illegal. Most research on applying ECs to the IPPS problem devised either complicated genetic operators or extra repairing algorithms to guarantee the feasibility or legality of offspring in the evolutionary process. This makes EC-based approaches for the IPPS problem more complex and may reduce computing performance in terms of the solution quality and the computing speed.

This paper considers a Flexible Job-shop Scheduling Problem with Operation and Processing flexibility (FJSP-OP, Doh et al. [17] referred to this problem as FJSP-MPP), a common and practical case of the IPPS problem, in which each job has multiple process plans resulting from operation and processing flexibility. The FJSP-OP has been investigated by some researchers, including Baykasoglu [4], Baykasoglu et al. [5], Shao et al. [58], Özgüven et al. [51], Rajabinasab and Mansour [56], Doh et al. [17], Zhang et al. [72], Liu et al. [43] and Luo et al. [44]. To solve this problem and overcome the limitations of current encoding strategies reported in the literature, this paper first presents a new representation scheme called the Four-Tuple Scheme (FTS). The FTS models a job with operation and processing flexibility, whereby alternative process plans for a job can easily be generated. Then, an enhanced GA with an innovative encoding strategy is developed for solving the FJSP-OP, with the objective of minimizing the makespan of all jobs. The merit of the proposed encoding strategy is that it can adopt the classic genetic operators
to the utmost extent without generating infeasible offspring and thus may improve
the solution quality and the computing speed. This is proven by the computational
experiments conducted in this paper.

The remainder of the paper is structured as follows. The next section provides a
relevant literature review on EC’s encoding strategy for the IPPS problem. Section
3 describes the FJSP-OP. Section 4 proposes the FTS to model jobs with operation
and processing flexibility. Section 5 develops a GA-based approach for the FJSP-
OP. Experimental studies and discussions are reported in Section 6, and conclusions
are summarized in the final section.

2. Literature review. Since Chryssolouris and Chan [13] first proposed the
preliminary idea of IPPS, numerous efforts have been devoted to solving the IPPS
problem. Three primary integration models have been proposed: Non-Linear
Approach (NLA), Closed Loop Approach (CLA) and Distributed Approach (DA) [5].
Reviews of the IPPS literature are available in previous publications [58, 54] and
not repeated here. This paper mainly focuses on encoding strategies of EC-based
approaches for the IPPS problem. Encoding strategies for the IPPS problem are
closely related to the representation of these three types of flexibility. This section
starts with introducing representations of flexibilities.

2.1. Representations of flexibilities. Two graph-based methods are extensively
adopted to represent the aforementioned three types of flexibility: network represen-
tation [25] and AND/OR graph [42, 32]. These two methods are similar in terms of
representing these flexibilities. Taking network representation as an example, there
are three node types in the network: starting node, intermediate node and ending
node [31]. The starting node and the ending node labelled ‘S’ and ‘E’ respectively,
are dummy nodes that denote the beginning and the completion of a job, respec-
tively. An intermediate node represents an operation, which contains alternative
machines used to perform the operation and the corresponding processing time re-
quired. An arrow connecting two nodes represents the precedence between these
two nodes. OR relationships are used to describe processing flexibility that the same
manufacturing feature can be completed with different operation procedures. If the
links following a node are connected by an OR symbol (called OR-connector), only
one of the OR-links (the links connected by an OR-connector) must be traversed.
OR-link path is an operation path that begins at an OR-link and ends as it merges
with the other paths, and its end is denoted by a JOIN-connector [31]. For the links
that are not connected by OR-connectors, all of them must be visited.

Figure 1 presents an example containing three jobs (job 1, 2 and 3) with operation
and processing flexibility. In the network of Figure 1(a), paths \{2,3,4\}, \{5,6,7\} and
\{5,8\} are three OR-link paths. An OR-link path can definitely contain other OR-
link paths, e.g., paths \{6,7\} and \{8\}.

However, many prior research assumed that an OR relationship contains only
two OR-link paths [31, 62, 58]. This is unrealistic and not applicable when an OR-
connector contains more than two OR-link paths. For example, the OR relationship
\text{OR}_{32} of job 3 in Figure 1(c) has three OR-link paths: \{2,3\}, \{4,5\} and \{6\}. Even
though, an OR relationship with more than two OR-link paths can be rewritten
into a new form in which each OR relationship contains only two OR-link paths.
As shown in Figure 1(d), the OR relationship \text{OR}_{32} can be rewritten into two new
OR relationships both with only two OR-link paths. Obviously, this will increase
the complexity of the network.
2.2. Encoding strategies for the IPPS problem. The vast majority of ECs adopt an encoding/decoding strategy with which genes on chromosomes are represented by numbers or letters [46]. Commonly used encoding approaches include binary encoding, permutation encoding, real number encoding, and general data structure encoding [22]. Encoding strategy not only plays a critical role in representing a problem but also ultimately affects the selection or design of the genetic operators.

Since Davis [16] made one of the first attempts to use GAs, as well as to design encoding/decoding strategy, for the conventional Job-shop Scheduling Problem (JSP), many encoding/decoding strategies have been proposed for the JSP, the conventional Flexible Job-shop Scheduling Problem (FJSP, FJSP is an extension of the JSP by allowing an operation to be performed on more than one machine), and the IPPS problem. For example, nine encoding strategies, such as operation-based representation, job-based representation and job pair relation-based representation, have been proposed for the JSP [11]. Among these encoding strategies, operation-based representation, one type of permutation encoding, is extensively adopted since it is the only one which keeps both information of operation sequence and precedence constraints in a chromosome [21, 12]. For an n-job and m-machine JSP, operation-based representation encodes a schedule as a fixed-length chromosome containing

**Figure 1.** Alternative process plans networks

- (a) job1
- (b) job2
- (c) job3
- (d) rewriting of OR32
\[ \sum_{i=1}^{n} n_i \] genes. Each symbol (commonly in the form of job number, hereafter use ‘job number’ directly) of a job appears in the chromosome exactly \( n_i \) times to represent the \( n_i \) ordered operations of this job and the \( k \)-th appearance of a job number from left to right stands for the \( k \)-th operation of this job. The advantage of operation-based representation is that a chromosome always corresponds to or can be decoded into a feasible schedule [8], and the classic mutation operators, such as inversion mutation, insertion mutation and reciprocal exchange mutation, are suitable for mutation operations without generating illegal offspring. However, the classic crossover operators, such as single-point crossover and two-point crossover, are not suitable for operation-based representation and will generate illegal offspring as these crossover operators cannot ensure that each job appears exactly \( n_i \) times in the chromosome. Therefore, many complicated crossover operators, such as partial-mapped crossover (PMX) [23], order-based crossover (OBX) [61], position-based crossover (PBX) [61], and precedence preservative crossover (PPX) [7], have been proposed for an operation-based representation. For more details about encoding strategies and genetic operators for the JSP, we refer readers to Cheng et al. [11, 12].

Later, Gao et al. [19] and Ho et al. [24] expanded the application of operation-based representation to the conventional FJSP. They split a chromosome into two independent substrings both with a fixed length \( \sum_{i=1}^{n} n_i \) : operation sequence substring and machine assignment substring. The first one is an operation-based representation standing for a list of ordered operations. The second one represents the machines selected for the operations of all jobs. This encoding strategy inherits the characteristics of the operation-based representation for the JSP. Moreover, since different operations may have different processing machine sets, most of the classic mutation operators, such as inversion mutation [26], insertion mutation [18], and reciprocal exchange mutation [3], are unsuitable for the machine assignment substring. Therefore, more complicated operators, such as permutation of two operations [27], have to be adopted or designed for the machine assignment substring.

For the IPPS problem, each job has multiple process plans, and the number of operations of the process plan selected for a job is not fixed. Therefore, it is hard to guarantee a fixed-length chromosome and thus leads to a great challenge of devising a suitable encoding strategy. Many efforts have been devoted to encoding strategies for the IPPS problem. This paper categorizes these existing encoding strategies into two types: hierarchical encoding approach and integrated encoding approach.

### 2.2.1. The hierarchical encoding approach

The hierarchical encoding approach encodes a chromosome as a fixed-length string(s), where some gene stands for a process plan or a processing route of a job. This means that process plans or routes for all jobs should be generated in advance. For example, Lee and Kim [35] proposed a simulation-based genetic algorithm for the IPPS problem. First, they encoded a chromosome as a fixed-length string where the \( i \)-th gene represents the index of one route selected for the \( i \)-th job. This means that a chromosome is a combination of routes of all jobs, and thus stands for a standard JSP. Then, a simulation-based scheduling model is used to obtain the scheduling performance measure (e.g. makespan) of each chromosome, and the performance measure is fed into a genetic algorithm in order to improve the solution quality. Later, Shao et al. [58] employed a two-substring based encoding strategy to represent a solution for the IPPS problem. The first substring, encoded by operation-based representation is the operation sequence substring with a fixed length \( n \times q \), where \( q \) is the number of operations of
the route which has the most operations among all the alternative routes of \( n \) jobs to be scheduled. If the number of all the operations of all routes selected for all jobs, \( s \), is not equal to \( n \times q \), \( n \times q - s \) genes in the operation sequence substring are randomly filled with 0, where a 0 stands for a dummy operation to be processed on any machine with zero processing time. The second substring is the route substring. The positions from 1 to \( n \) in this substring represent the job from 1 to \( n \), and the number in the \( i \)-th position represents the index of a route selected for the \( i \)-th job. In addition, there are some variants of the encoding strategy of Shao et al. [58], such as Qiao and Lv [55] and Li et al. [40, 38, 39]. For example, Qiao and Lv [55] combined these two substrings into one string. Each gene in a chromosome is made up of two positive integer numbers. The first one is the index of a route selected for the job represented by the second number in the same gene, and the second one is the job number of a job.

Figure 2 shows an illustrative chromosome for the problem shown in Figure 1 by applying the encoding strategy of Shao et al. [58]. Some alternative process routes for the three jobs are listed in Figure 2(a). Each element of the process routes takes the form of \((i, m)\), meaning the \( i \)-th operation of its corresponding job is processed on machine \( m \). It follows that \( q \) is equal to 5 and thus the length of the operation sequence substring is equal to 15. The first number, i.e. 3, in the route substring indicates that the third process route is selected for job 1. Since the third process route of job 1 contains four operations, the number of job 1 appears four times in the operation sequence substring.

![Figure 2](image)

**Figure 2.** An illustrative chromosome applying the hierarchical encoding approach

The shortcomings of the hierarchical encoding approach lie in: (1) process plans or routes for a job should be generated in advance, but for a job with high production flexibility, generating several best or all process plans or routes will result in a combinational-explosive problem; (2) an EC adopting the hierarchical encoding approach cannot simultaneously find a suitable process plan for each job and an optimal or near optimal schedule; and (3) it follows that almost all the classic genetic operators are unsuitable for the hierarchical encoding approach, and thus dedicated genetic operators have to be designed to cope with this type of encoding strategy. For instance, Shao et al. [58] designed a complicated crossover operator for the operation sequence substring by appending or deleting 0 to ensure that \( n \times q - s \) genes in this substring are 0. This may reduce the optimization performance of their algorithm.
2.2.2. The integrated encoding approach. The integrated encoding approach intends to simultaneously represent the process plans selected for all jobs and a feasible schedule for the IPPS problem. For example, Kim et al. [31] proposed an SEA-based approach for the IPPS problem. In this approach, two types of different populations are constructed and maintained during evolution: process planning populations ($Pop[k], k = 1, \ldots, n$) and scheduling population ($Pop[n+1]$). $Pop[k]$ ($k = 1, \ldots, n$) consists of individuals representing process plans for job $k$, and $Pop[n+1]$ consists of individuals representing schedules for $n$ jobs to be scheduled. Each individual in population $Pop[k]$ ($k = 1, \ldots, n$) consists of three substrings: head substring, tail substring and machine assignment substring. The head substring represents all the possible operations for the $k$-th job in a sequential list. The tail substring encodes OR relationships in the $k$-th job as the binary numbers of 0 and 1, where the number 0 and the number 1 represent selecting the right and the left OR-link path of an OR relationship, respectively. The head substring and the tail substring can be decoded into a process plan for this job. The machine assignment substring is responsible for machine assignments, the $i$-th element of which represents the machine on which the operation corresponding to the $i$-th element of the head substring is processed. This means the machine assignment substring depends on the head substring. An individual in $Pop[n+1]$ is a permutation of the operations for all jobs. This encoding strategy can guarantee fixed-lengths for all the substrings. However, the authors tried to avoid generating infeasible offspring and ensure the $i$-th element of the machine assignment substring stands for the machine to process the operation represented by the $i$-th element of the head substring. To achieve this, the authors presented a variation of the classic two-point crossover both and simultaneously for crossover operations of the head and the machine assignment substrings. The encoding method of Kim et al. [31] has been widely employed in the domain of the IPPS problem. Examples include Wang and Ju [62] and Zhang et al. [72].

Considering the problem shown in Figure 1 again, Figure 3 shows illustrative chromosomes by applying the encoding strategy of Kim et al. [31]. A chromosome in $Pop[1]$ is shown in Figure 3(a), representing the process plan for job 1 (the other chromosomes in $Pop[2]$ and $Pop[3]$ are similar to that in $Pop[1]$). Figure 3(b) shows a chromosome in $Pop[n+1]$ with a length of 28, indicating that the problem has 28 operations in total. Operation numbers of 10-18 and 19-28 in Figure 3(b) correspond to those of 1-9 in job 2 and those of 1-10 in job 3, respectively.

![Illustrative chromosomes applying the integrated encoding approach](image-url)
Compared to the hierarchical encoding approach, the integrated encoding approach can overcome the combinational-explosive problem of generating all alternative process plans. However, it also faces the same problem as that of the hierarchical encoding approach, that is, the application of the majority of the classic genetic operators is restricted and dedicated genetic operators have to be designed.

As mentioned previously, the current encoding strategies for the IPPS problem are very complicated and unsuitable for applying the classic genetic operators. Therefore, what is expedient for solving the FJSP-OP by using EC is to design an efficient encoding strategy that can simplify the genetic operations, and avoid generating illegal chromosomes and/or offspring. In this paper, we employ a genetic algorithm and propose an innovative encoding strategy for the FJSP-OP. The proposed encoding strategy can adopt the classic genetic operators to the utmost extent.

3. Problem statement. The FJSP-OP considered in this paper may be briefly expressed as follows. There is a set of \( n \) independent jobs, \( \text{Jobs} = \{J_1, \ldots, J_n\} \), to be processed on a set of \( m \) machines, \( \text{Machines} = \{M_1, \ldots, M_m\} \). Each job has multiple process plans and multiple processing routes due to operation and processing flexibility. The aim of the FJSP-OP is to select a suitable process plan for each job (process plan selection sub-problem), assign each operation contained in the selected process plan to a machine (routing sub-problem), and order all operations selected for all jobs on the machines (scheduling sub-problem) in such a manner that some relevant criteria, e.g. minimum makespan and minimum mean flow time, are achieved. In this paper, one objective, i.e. minimizing makespan, is considered. In solving this problem, the following assumptions are made:

   (1) All jobs are ready for processing at time zero.
   (2) Job pre-emption is not allowed and each machine can process only one operation at a time.
   (3) Setup times for the jobs are sequence-independent and can thus be included in the processing times.

Several papers have systematically examined mathematical models for the IPPS problem [51, 28], as well as the FJSP-OP [17]. For the sake of simplicity, this paper does not present a detail mathematical model for the FJSP-OP. Interested readers may refer to Doh et al. [17] and Jin et al. [28].

4. A Four-Tuple Scheme for representing a job with operation and processing flexibility. In this section, the classic network representation for the FJSP-OP is rewritten into a new mathematical model, i.e., Four-Tuple Scheme (FTS).

An FTS for modelling a job \( J \) with operation and processing flexibility is defined as follows:

\[
J = \{O_S, M_S, OR_S, Map\}
\] (1)

where \( O_S = \{O_1, \ldots, O_n\} \) represents a set of all the operations of \( J \); \( M_S = \{M(O_1), \ldots, M(O_n)\} \) is a set of processing machines for all the operations in which each element \( M(O_i) \) consists of pairs \((m, p)\) indicating that operation \( O_i \) can be processed on machine \( m \) for \( p \) time units; \( ORs = \{OR_1, \ldots, OR_r\} \) is a set of all the OR relationships of \( J \); and \( Map \) is an ordered set of operation(s) and OR relationship(s) describing the general processing order of \( J \).

An OR relationship \( OR \) is a set of OR-link paths \( OR = \{Path_i | i = 1, \ldots, p\} \) and an OR-link path \( Path_i \) is an ordered set of operation(s) and/or OR relationship(s),
dictating that these operation(s) and/or OR relationship(s) are processed based on the priority of their appearance in Path. For the sake of distinction, any OR relationship must be assigned a unique name (or label). If an OR relationship is one element of an OR-link path, it is referred to as ‘embedded OR relationship’. An embedded OR relationship must appear in the form of its unique name in this OR-link path.

It is worth mentioning that any OR relationship, even the one embedded into an OR-link path, should be an element of $OR_S$, while any embedded OR relationship should not be contained in $Map$.

Take job 1 as an example (see Figure 1(a)). We have $O_S = \{O_{11}, \ldots, O_{19}\}$, $M_S = \{M(O_{11}), \ldots, M(O_{19})\}$, $OR_S = \{OR_{11}, OR_{12}\}$ and $Map = \{O_{11}, OR_{11}, O_{19}\}$, where $OR_{11} = \{\{O_{12}, O_{13}, O_{14}\}, \{O_{15}, OR_{12}\}\}$, $OR_{12} = \{\{O_{16}, O_{17}\}, \{O_{18}\}\}$, and $M(O_{11}) = \{(2, 18), (4, 22)\}$ (processing machines for other operations can be obtained from Figure 1). $Map = \{O_{11}, OR_{11}, O_{19}\}$ indicates that the general processing order of job 1 is $O_{11} - OR_{11} - O_{19}$. The OR relationship $OR_{12}$ does not appear in $Map$ as it is an embedded OR relationship.

The FTS is useful for generating alternative process plans for a job. This paper presents a recursive algorithm named GENERATE-PROCESS-PLAN to generate alternative process plans for job $J = \{O_S, M_S, OR_S, Map\}$. To begin with, we should input an array $ORArr = \{b_1, \ldots, b_r\}$ (called input array), in which each element $b_i$, called the index number of OR-link path, is an integer in the range of $[1, |OR|R|$ (the number of OR-link paths of an OR relationship OR is denoted by $|OR|R$), and indicates that the $b_i$-th OR-link path of $OR_i$ will be traversed or selected. Next, an initial processing order is obtained by $Map$. When the processing order contains names of OR relationships, any name of OR relationship should be replaced by the OR-link path of its corresponding OR relationship specified by the input array $ORArr$. The algorithm GENERATE-PROCESS-PLAN is outlined in Figure 4. For example, the initial processing order for job 1 (see Figure 1(a)) is $O_{11} - OR_{11} - O_{19}$. It obviously contains a name of an OR relationship, $OR_{11}$.

\begin{figure}[h]
\begin{center}
\begin{algorithm}
\textbf{Input:} $ORArr = \{b_1, \ldots, b_r\}$ // specify an OR-link path to be traversed for each OR relationship  
\textbf{Output:} $ProcessPlan$ // a feasible process plan  
\textbf{Procedure:}  
\begin{itemize}
  \item obtain the initial processing order $ProcessPlan$ by $Map$;  
  \item while ($ProcessPlan$ contains a name of an OR relationship $OR\_relation$)  
    \begin{itemize}
      \item obtain the index $i$ of the name in $ProcessPlan$;  
      \item obtain the index number of OR-link path, $b$, of $OR\_relation$ by $ORArr$;  
      \item delete the $i$-th element in $ProcessPlan$, and insert the $b$-th OR-link path of $OR\_relation$ into $ProcessPlan$ at the same position;  
    \end{itemize}
\end{itemize}
\textbf{end while};  
\begin{itemize}
  \item output $ProcessPlan$.  
\end{itemize}
\end{algorithm}
\end{center}
\caption{Algorithm GENERATE-PROCESS-PLAN}
\end{figure}

Suppose that $ORArr = \{1, 1\}$ indicating that the first OR-link path of $OR_{11}$ will be selected. Then, $OR_{11}$ in $O_{11} - OR_{11} - O_{19}$ is replaced by its first OR-link path, i.e., \{O_{12}, O_{13}, O_{14}\}, and the processing order is updated into $O_{11} - O_{12} - O_{13} - O_{14} - O_{19}$.
At this point, the processing order does not contain any name of OR relationship and thus is a process plan generated for job 1.

We can randomly generate different input arrays, that is, each element \( b_i \) of \( ORArr \) should be randomly generated in the range of \([1, |OR_i|]\). Then, applying this algorithm, we will obtain all the alternative process plans for a job. For example, there are, in total, three alternative process plans for job 1 as follows:

1. \( O_{11} - O_{12} - O_{13} - O_{14} - O_{19} \), if \( ORArr = \{1, 1\} \) or \( ORArr = \{1, 2\} \).
2. \( O_{11} - O_{15} - O_{16} - O_{17} - O_{19} \), if \( ORArr = \{2, 1\} \).
3. \( O_{11} - O_{15} - O_{18} - O_{19} \), if \( ORArr = \{2, 2\} \).

It is worth highlighting that different input arrays may generate a same process plan. For example, both input arrays \( \{1, 1\} \) and \( \{1, 2\} \) result in the same process plan, \( O_{11} - O_{12} - O_{13} - O_{14} - O_{19} \), for job 1. This is because the first elements of these two input arrays assert that \( OR_{11} \) in \( Map \) of job 1, \( O_{11} = OR_{11} - O_{19} \), will be replaced by the OR-link path \( \{O_{12}, O_{13}, O_{14}\} \), and thus the second elements of these two input arrays will not affect the final process plan of job 1.

Obviously, the FTS is precise and concise in terms of formulation of modelling a job with operation and processing flexibility and can easily generate all the alternative process plans for a job by randomly generating different input arrays.

5. A GA-based approach. To solve the FJSP-OP, this paper proposes a GA-based approach. The approach adopts a simple but effective three-substring based encoding/decoding strategy, where a chromosome consists of three substrings: OR substring, machine assignment substring and operation sequence substring. The OR substring specifies an alternative process plan for each job, the machine assignment substring assigns only one machine for each operation of all jobs, and the operation sequence substring sequences the relative operations determined by the OR substring on a set of machines determined by the machine assignment substring.

5.1. The maximum numbers of OR-link paths and of machines. Suppose that in the FJSP-OP, each job \( J_i \) \((i = 1, \ldots, n)\) is denoted by \( J_i = \{O_S(i), M_S(i), OR_S(i), Map(i)\} \), where \( O_S(i) = \{O_{i1}, \ldots, O_{in(i)}\} \), \( M_S(i) = \{M(O_{i1}), \ldots, M(O_{in(i)})\} \), and \( OR_S(i) = \{OR_{i1}, \ldots, OR_{ir(i)}\} \). In addition, \( r(i) \) and \( n(i) \) represent the number of the OR relationships and the number of all the operations of \( J_i \), respectively.

The maximum number of OR-link paths \( maxP \) is equal to the number of all the OR-link paths of the OR relationship that has the most OR-link paths among all the OR relationships of all jobs, i.e.,

\[
maxP = \max\{|OR_{ij}| i = 1, \ldots, n, j = 1, \ldots, r(n)\}
\] (2)

The maximum number of machines \( maxM \) is equal to the number of all the alternative machines for the operation that has the most alternative machines among all the operations of all jobs, i.e.,

\[
maxM = \max\{|M_{ij}| i = 1, \ldots, n, j = 1, \ldots, n(n)\}
\] (3)

These two variables \( maxP \) and \( maxM \) are used for encoding and decoding the OR substring and the machine assignment substring, respectively.

5.2. Encoding strategy. Each chromosome consists of three substrings: OR substring, machine assignment substring and operation sequence substring.

(1) The OR substring
The OR substring is responsible for selections of the OR-link path for all the OR relationships of all jobs. It is \(n\)-partitioned by \(jobs\) as follows:

\[
\begin{align*}
&b'_{i1}, \ldots, b'_{i,r(i)} \\
&J_i
\end{align*}
\]

where each element \(b'_{ij}\) is a random integer in \([1, \text{maxP}]\). In this substring, each job \(J_i\) has \(r(i)\) numbers, i.e., \(b'_{i1}, \ldots, b'_{i,r(i)}\), and the length of the OR substring is equal to \(\sum_{i=1}^{n} r(i)\).

(2) The machine assignment substring

The machine assignment substring is responsible for machine assignments for all the operations of all jobs. It is also \(n\)-partitioned by \(jobs\). Details of this substring are as follows:

\[
\begin{align*}
&m'_{i1}, \ldots, m'_{i,n(1)} \\
&J_i
\end{align*}
\]

where each element \(m'_{ij}\) is a random integer in \([1, \text{maxM}]\). In this substring, each job \(J_i\) has \(n(i)\) numbers, i.e., \(m'_{i1}, \ldots, m'_{i,n(i)}\), and the length of the machine substring is equal to \(\sum_{i=1}^{n} n(i)\).

(3) The operation sequence substring

The operation sequence substring is an operation-based representation. It includes all the possible operations for each job and has the same length as the machine assignment substring.

Figure 5 is an example of the chromosome for job 1, 2 and 3 as shown in Figure 1. In this example, \(\text{maxP} = 3\) and \(\text{maxM} = 3\).

5.3. Decoding strategy. The procedure for decoding a chromosome is as follows: The first step is to decode the OR substring and the machine assignment substring of a chromosome into alternative process plans for all jobs and alternative machines for all the operations of all jobs, respectively. Then the results of the first step are used as input data for decoding the operation sequence substring into a feasible schedule.

5.3.1. Decoding the OR substring. The OR substring of a chromosome can be decoded into a set of process plans for all jobs as follows. First, for any job \(J_i\), we obtain the number \(b'_{ij}\) of any OR relationship \(OR_{ij}\) from the OR substring. Then, \(b'_{ij}\) will be mapped to an exact OR-link path of \(OR_{ij}\) using the following formula:

\[
b_{ij} = \left\lceil \frac{b'_{ij}}{\text{maxP}} \times |OR_{ij}| \right\rceil \quad i = 1, \ldots, n, j = 1, \ldots, r(i)
\]
where \([x]\) is the smallest integer greater than or equal to \(x\), and \(b_{ij}\) means that the \(b_{ij}\)-th OR-link path of \(OR_{ij}\) is selected.

We claim that \(b_{ij} \in [1,|OR_{ij}|]\). Since \(b'_{ij}\) is an integer in \([1,\max\{P\}]\), there exists \(\frac{b'_{ij}}{\max\{P\}} \in (0,1]\). This implies that \(\frac{b'_{ij}}{\max\{P\}} \times |OR_{ij}| \in (0,|OR_{ij}|]\), and then \(\left\lfloor \frac{b'_{ij}}{\max\{P\}} \times |OR_{ij}| \right\rfloor \in [1,|OR_{ij}|]\), i.e., \(b_{ij} \in [1,|OR_{ij}|]\). It is worth emphasising that \(b_{ij} \in [1,|OR_{ij}|]\) ensures that we always get a valid OR-link path for an OR relationship by applying formula (4).

Finally, all the numbers \(b_{ij}(i=1,\ldots,n, j=1,\ldots,r(i))\) are used to construct the input array of the algorithm GENERATE-PROCESS-PLAN for job \(J_i(i=1,\ldots,n)\). Then this algorithm generates a set of process plans for all jobs.

Take the OR substring, shown in Figure 5, as an example. The number \(b'_{11}\) related to \(OR_{11}\) is 3, and we have \(b_{11} = 2\) by applying formula (4), which indicates that the second OR-link path, i.e., \(\{O_{15},OR_{12}\}\), is selected for \(R_{11}\) (\(OR_{11} = \{\{O_{12},O_{13},O_{14}\},\{O_{15},OR_{12}\}\}\)). Similarly, we have the other numbers: \(b_{12} = 1, b'_{21} = 2, b_{22} = 1, b_{23} = 2, b_{31} = 1\) and \(b_{32} = 2\). Therefore, the input array for job 1 is \(\{2,1\}\), that for job 2 is \(\{2,1,2\}\), and that for job 3 is \(\{1,2\}\). The final decoded process plans for these three jobs are listed as follows (see Figure 7):

- Process plan for job 1: \(O_{11} - O_{15} - O_{16} - O_{17} - O_{19}\).
- Process plan for job 2: \(O_{21} - O_{24} - O_{28} - O_{29}\).
- Process plan for job 3: \(O_{37} - O_{38} - O_{39} - O_{3,10}\).

5.3.2. Decoding the machine assignment substring. The machine assignment substring of a chromosome can be decoded into a set of machines for all the operations of all jobs. First, for any job \(J_i\), we obtain the number \(m'_{ij}\) for \(O_{ij}\) from the machine assignment substring. Then, \(m'_{ij}\) will be mapped to an alternative machine for processing \(O_{ij}\) using the following formula:

\[
m'_{ij} = \left\lfloor \frac{m'_{ij}}{\max\{M\}} \times |M(O_{ij})| \right\rfloor , \quad i=1,\ldots,n, j=1,\ldots,r(i)
\]  

where \(m_{ij}\) means that \(O_{ij}\) is assigned to the \(m_{ij}\)-th machine (or element) of set \(M(O_{ij})\). We also claim that \(m_{ij} \in [1,|M(O_{ij})|]\). The proof is similar to that of \(b_{ij} \in [1,|OR_{ij}|]\), \(m_{ij} \in [1,|M(O_{ij})|]\) ensures that each operation is always assigned to a valid machine by applying formula (5).

The chromosome shown in Figure 5 is again used as an example. The number \(m'_{12}\) related to \(O_{12}\) is 3, and we have \(m_{12} = 2\) by applying formula (5), which means that \(O_{12}\) is assigned to the second alternative machine, i.e., machine 8, in \(M(O_{12})\) \(\{\{7,39\},\{8,36\}\}\). The final decoded alternative machines for processing all the operations of the three jobs are listed as follows:

- The machine numbers for performing all the operations for job 1 from \(O_{11}\) to \(O_{19}\) are \(\{4,8,1,6,5,2,5,7,8\}\).
- The machine numbers for performing all the operations for job 2 from \(O_{21}\) to \(O_{29}\) are \(\{4,8,2,8,5,2,6,7,8\}\).
- The machine numbers for performing all the operations for job 3 from \(O_{31}\) to \(O_{3,10}\) are \(\{5,6,2,7,8,5,1,7,8,4\}\).

5.3.3. Decoding the operation sequence substring. Once the process plans for all jobs, \(\{ProcessPlan(i)|i=1,\ldots,n\}\), and the machines to perform all the operations of all jobs, \(\{Machine(O_{ij})|i=1,\ldots,n, j=1,\ldots,r(i)\}\), are obtained, the
operation sequence substring of a chromosome can be decoded into an operation
precedence representing a feasible schedule for the FJSP-OP by an algorithm called
GENERATE-SCHEDULING. Both \{ProcessPlan(i)|i = 1,\ldots,n\} and \{Machine
(O_{ij})|i = 1,\ldots,n, j = 1,\ldots,n(i)\} are used as input data for this algorithm. Be-
cause the operation sequence substring includes all the possible operations for each
job but the process plan selected for each job may not include all the possible oper-
ations of the job, we delete \( n(i) - |ProcessPlan(i)| \) extra elements labelled \( i \)
from right to left for each job \( J_i \) in this algorithm, where \( |ProcessPlan(i)| \)
stands for the number of operations of the process plan selected for job \( J_i \). Then, for
the remainder of the operation sequence substring, the number of each job \( J_i \)
appears exactly \( |ProcessPlan(i)| \) times.

The output of the GENERATE-SCHEDULING is an operation precedence rep-
resenting a feasible schedule for the FJSP-OP. An operation precedence is a list
of ordered triples \((i, j, k)\), one for an operation, which stands for the operation \( O_{ij} \)
is processed on machine \( k \).

The procedure of the algorithm GENERATE-SCHEDULING is outlined in Fig-
ure 6.

\begin{figure}[h]
\centering
\begin{algorithm}
\SetAlgoLined
Input: \{ProcessPlan(i)|i = 1,\ldots,n\} and \{Machine(O_{ij})|i = 1,\ldots,n,
\hspace{1em} j = 1,\ldots,n(i)\}
Output: OperationPrecedence //an operation precedence
Procedure:
\hspace{1em}set OperationPrecedence = \emptyset;\\
\hspace{1em}set count[i] = 0, i = 1,\ldots,n;\\
\hspace{1em}for job \( J_i \) in the set of jobs\\
\hspace{2em}delete \( n(i) - |ProcessPlan(i)| \) elements labelled \( i \) from right to left in\\
\hspace{2em}the operation sequence substring;\\
\hspace{1em}end for;\\
\hspace{1em}for each element \( i \) in the operation sequence substring\\
\hspace{2em}set count[i] = count[i] + 1; // each element \( i \) represents a job number;\\
\hspace{2em}set curOperation be the count[i]-th operation of ProcessPlan(i);\\
\hspace{2em}look for the operation number of curOperation, \( j \), in set \( OS(i) \);\\
\hspace{2em}look for the machine number of curOperation, \( k \), in set \( Machine(O_{ij}) \);\\
\hspace{2em}append \((i, j, k)\) to the end of OperationPrecedence;\\
\hspace{1em}end for;\\
output OperationPrecedence;
\end{algorithm}
\caption{Algorithm GENERATE-SCHEDULING}
\end{figure}

This decoding scheme is illustrated by the chromosome shown in Figure 7. Since
the process plan for job 1 determined by the OR substring, \( O_{11} - O_{15} - O_{16} - O_{17} -
O_{19} \), contains 5 operations and job 1 has 9 operations, we delete four 1s from right
to left in the operation sequence substring. Similarly, five 2s for job 2 and six 3s
for job 3 are deleted. The resulting operation precedence is \((1,1,4)-(2,1,4)-
(3,7,1)-(3,8,7)-(2,4,8)-(2,8,7)-(1,5,5)-(3,9,8)-(1,6,2)-(2,9,8)-(3,10,4)-(1,7,5)-(1,9,8)\).

An operation precedence can be decoded into semi-active, active, non-delay, and
hybrid schedules [8]. The active schedule is adopted in this study. For the sake
of simplicity, this paper does not present details of the active schedule. Interested readers may refer to Baker [2].

Clearly, the proposed encoding and decoding strategies closely incorporate the FTS. A chromosome generated by this encoding strategy always represents a feasible solution. The three sub-problems of the FJSP-OP are simultaneously solved by decoding a chromosome, that is, the OR substring, the machine assignment substring and the operation sequence substring give answers to the process plan selection sub-problem, the routing sub-problem and the scheduling sub-problem, respectively. Furthermore, these three substrings are independent in terms of the encoding rules and the representing meaning. The independence of these three substrings will bring potential advantages in freely designing or selecting genetic operators for each substring.

5.4. Initial population and fitness evaluation. In order to operate evolutionary algorithm, an initial population is necessary. The generation of the initial population in GA is done randomly based on the encoding strategy as discussed in Section 5.2.

This study considers only one objective function for the FJSP-OP, i.e., minimizing the makespan. Thus the makespan is used to evaluate fitness. Here we employ the well-known roulette wheel selection along with the elitist strategy for the reproduction operation.

5.5. Crossover. Due to the independence of the three substrings in a chromosome, this paper selects three independent classic crossover operators for the three substrings of chromosomes. The first and the second one adopt single-point crossover for both the OR substrings and the machine assignment substrings (actually any one of the classic crossover operators can be selected for these two substrings). The third one adopts precedence preserving order-based crossover (POX) for the operation sequence substrings. POX is also a classic crossover operator for the
operation-based representation. In the evolution procedure, each crossover operator only exchanges information contained in its target substrings and is chosen randomly in every generation. Figure 8 is an example where the three crossover operators are all chosen.

The procedure of single-point crossover is described as follows [26]: First, randomly select two parent chromosomes $P_1$ and $P_2$, and initialize two empty offspring $O_1$ and $O_2$. Next, randomly select a crossover point. Then, copy the elements of $P_1$ and $P_2$, which appear in the left side of the crossover point, to the same positions in $O_1$ and $O_2$, respectively; the elements of $P_1$ and $P_2$, which are in the right side of the crossover point, are appended to the same positions in $O_2$ and $O_1$, respectively.

The procedure of POX is described as follows [36]: First, randomly select two parent chromosomes $P_1$ and $P_2$, and initialize two empty offspring $O_1$ and $O_2$. Next, the set of job numbers is randomly divided into two nonempty subsets $S_1$ and $S_2$. Then, each element of $P_1$ that belongs to $S_1$ is copied to $O_1$ at the same positions, and each element of $P_2$ that belongs to $S_2$ is appended to the remaining position of $O_1$ in the same order that they appear in $P_2$. Another offspring ($O_2$) can be produced by switching the roles of $P_1$ and $P_2$.

5.6. Mutation. For the same reason as selecting the crossover operators mentioned in Section 5.5, this paper selects three independent classic mutation operators. The first one adopts one-point mutation for the OR substrings, the second one adopts inversion mutation for the machine assignment substrings, and the third one adopts reciprocal exchange mutation for the operation sequence substrings. In the evolution procedure, each mutation operator only changes information contained in its target substrings and is chosen randomly in every generation. Figure 9 is an example where the three mutation operators are all chosen.

The procedure of one-point mutation is described as follows [26]: First, randomly select one chromosome. Then randomly select a point in the OR substring and change the number of this selected element to another integer number in $[1, \text{maxP}]$.

The procedure of inversion mutation is described as follows [26]: First, randomly select one chromosome. Then randomly select two points in the machine assignment substring and reverse the numbers between these two points.

The procedure of reciprocal exchange mutation is described as follows [3]: First, randomly select one chromosome. Then randomly select two points in the operation sequence substring and interchange these two numbers.

Because of the advantages resulting from the encoding and decoding strategies, the classic genetic operators can be directly implemented, and the offspring always represent feasible solutions.

6. Computational experiments. The proposed GA-based approach for the FJSP-OP is coded in C# language and implemented on a desktop computer with an Intel i7 2.0 GHz processor. To verify this approach, six computational experiments have been conducted.

6.1. Experiments.

6.1.1. Experiment 1. Experiment 1 is conducted for the problem shown in Figure 1. In this experiment, each job has both operation and processing flexibilities, and thus can be modelled by the FTS. The parameters for the proposed approach are as follows: Population size ($PopSize$) is 50, the probability of crossover ($P_c$) is 0.8,
the probability of mutation ($P_m$) is 0.1 and the iteration number ($IteNo$) is 50. The makespan of this problem is 121.

To demonstrate the facticity of the experimental results, Figure 10 illustrates the Gantt chart of Experiment 1. The number on each color bar represents the index of an operation of its corresponding job.

6.1.2. **Experiment 2.** Experiment 2, constructed with 5 jobs and 5 machines, is adopted from Moon et al. [47]. In this experiment, each job has multiple process plans resulting from operation and sequencing flexibility. Each job can be regarded as a job with only one OR relationship, in which any OR-link path represents a different process plan for this job. Therefore, all jobs can be modelled by the FTS and then this experiment can be conducted. The parameters for the proposed approach are as follows: $PopSize = 50$, $P_c = 0.8$, $P_m = 0.1$, and $IteNo = 50$. In
(a) The mutation operation for the OR substring

(b) The mutation operation for the machine assignment substring

(c) The mutation operation for the operation sequence substring

Figure 9. Mutation operations

Figure 10. The Gantt chart of Experiment 1

Table 1, the results of the proposed approach (GA-OP) are compared with those of Evolutionary Algorithm (EA) by Moon et al. [47], Modified GA by Shao et al. [58], and Priority-Based Heuristic Algorithm (PBHA) by Dabney et al. [15]. As depicted in Table 1, GA-OP reaches the best solution found to date.

Table 1. Comparison of results for Experiment 1

| Solution approach | EA   | Modified GA | PBHA | GA-OP |
|-------------------|------|-------------|------|-------|
| Makespan          | 16   | 14          | 14   | 14    |
| Mean CPU time (ms)| N/A  | N/A         | N/A  | 191   |

N/A means the result was not given by the author.

6.1.3. Experiment 3. Experiment 3 includes 6 jobs and 8 machines, and is taken from Shao et al. [58]. In this experiment, each job has both operation and processing flexibilities, and thus can be modelled by the FTS. In Table 2, the parameters and results of GA-OP are compared with those of Modified GA by Shao et al. [58] and the Ant Lion Optimization Algorithm (ALO) by Petrović et al. [53]. As depicted in Table 2, under the same conditions of crossover probability, mutation probability and iteration number, GA-OP with a dramatically reduced population
Table 2. Parameters and comparison of results on Experiment 3

| Solution approach | ALO | Modified GA | GA-OP |
|-------------------|-----|-------------|-------|
| Parameters        | N/A | Population = 500 | Population = 50 |
|                   |     | $P_c = 0.8, P_m = 0.1, IterNo = 100$ |     |
| Makespan          | 161 | 162         | 145   |
| Mean CPU time (ms)| N/A | N/A         | 309   |

Table 3. Parameters and comparison of results on Experiment 4

| Solution approach | ALGA | GA-OP |
|-------------------|------|-------|
| Parameters        | Population = 400 | Population = 200 |
|                   | $P_c = 0.8, P_m = 0.1, IterNo = 100$ |     |
| Makespan          | 188  | 181   |
| Mean CPU time (ms)| N/A  | 2,907 |

size significantly dominates the approaches of Modified GA and ALO. Figure 11 illustrates the Gantt chart of Experiment 3. For more details of the data and solution of Experiment 3, refer to the Appendix 1.

6.1.4. Experiment 4. Experiment 4 is provided by Li et al. [39] with 10 jobs and 6 machines. In this experiment, each job contains multiple process plans caused by operation and sequencing flexibility and can be modelled by employing the same method used for Experiment 2. In Table 3, the parameters and results of GA-OP are compared with those of Active Learning Genetic Algorithm (ALGA) by Li et al. [39]. Clearly, GA-OP with only half of the population of ALGA, obtains a better solution than ALGA does. Figure 12 illustrates the Gantt chart of Experiment 4.
6.1.5. Experiment 5. Experiment 5 is taken from Brandimarte’s data set (BRdata) [9] (https://www.ime.usp.br/~cris/fjs/benchmark/brandimarte/). It consists of ten famous benchmark problems. In this experiment, each job has only operation flexibility and thus can be modelled by the FTS. Table 4 shows the experimental results and comparisons with the other algorithms. In Table 4, $n \times m$ denotes the jobs and machines about each problem, and the results of GA-OP are compared with those of Two-stage Artificial Bee Colony (TABC) algorithm by Gao et al. [20], Hybrid Genetic Tabu Search (HGTS) by Palacios et al. [52], Memetic Algorithms2 (MA2) by Yuan and Xu [68], Hybrid Algorithm (HA) by Li and Gao [37], Particle Swarm Optimization (PSO) by Nouiri et al. [50] and Variable Neighborhood Search based Genetic Algorithm (VNSGA) by Zhang et al. [69]. Table 5 shows the computational time (in seconds) and the comparisons of the above algorithms. The parameters for GA-OP are as follows: $PopSize = 200$, $P_c = 0.8$, $P_m = 0.1$, and $IteNo = 200$. As depicted in Table 4, GA-OP reaches the best solutions found to date for all problems except for problem mk05 and mk10. The Gantt chart of problem mk06 is shown in Figure 13. One interesting point worth mentioning is that our Gantt chart of problem Mk06 is very different from that of Li and Gao [37], but both with the same makespan.

Table 4. Comparison of results for Experiment 5

| Problem | $n \times m$ | TABC | HGTS | MA2 | HA | PSO | VNSGA | GA-OP |
|---------|--------------|------|------|-----|----|-----|-------|-------|
| mk01    | 10x6         | 40   | 40   | 40  | 40 | 41  | 40    | 40    |
| mk02    | 10x6         | 26   | 26   | 26  | 26 | 26  | 26    | 26    |
| mk03    | 15x8         | 204  | 204  | 204 | 207| 204 | 204   |       |
| mk04    | 15x8         | 60   | 60   | 60  | 60 | 60  | 60    |       |
| mk05    | 15x4         | 173  | 172  | 172 | 171| 173 | 172   |       |
| mk06    | 10x15        | 60   | 57   | 59  | 57 | 57  | 57    | 57    |
| mk07    | 20x5         | 139  | 139  | 139 | 139| 137 | 139   |       |
| mk08    | 20x10        | 523  | 523  | 523 | 523| 523 | 523   |       |
| mk09    | 20x10        | 307  | 307  | 307 | 307| 307 | 307   |       |
| mk10    | 20x15        | 202  | 198  | 202 | 197| 312 | 198   | 198   |

Table 5. The experimental results (computational time in terms of seconds) of experiment 5

| Problem | $n \times m$ | TABC | HGTS | MA2 | HA | PSO | VNSGA | GA-OP |
|---------|--------------|------|------|-----|----|-----|-------|-------|
| mk01    | 10x6         | 3.36 | 5    | 20.16| 0.06| N/A | N/A   | 0.27  |
| mk02    | 10x6         | 3.72 | 15   | 28.21| 0.59| N/A | N/A   | 3.75  |
| mk03    | 15x8         | 1.56 | 2    | 53.76| 0.16| N/A | N/A   | 0.12  |
| mk04    | 15x8         | 66.58| 10   | 30.53| 0.49| N/A | N/A   | 3.41  |
| mk05    | 15x4         | 78.45| 18   | 36.36| 4.57| N/A | N/A   | 6.23  |
| mk06    | 10x15        | 173.98| 63   | 80.61| 53.82| N/A | N/A   | 65.14 |
| mk07    | 20x5         | 66.19| 33   | 37.74| 20.01| N/A | N/A   | 25.4  |
| mk08    | 20x10        | 2.15 | 3    | 77.71| 0.02| N/A | N/A   | 0.2   |
| mk09    | 20x10        | 304.43| 24   | 75.23| 0.86| N/A | N/A   | 4.1   |
| mk10    | 20x15        | 418.19| 104  | 90.75| 33.21| N/A | N/A   | 73.12 |
6.1.6. Experiment 6. Experiment 6, constructed with 18 jobs and 15 machines, is adopted from Kim et al. [31] (http://syslab.jnu.ac.kr/index.php?document_srl=528&mid=board_PDAS73) and has been investigated frequently in previous studies [40, 65]. In this experiment, each job has operation, sequencing and processing flexibility. Because the number of multiple process plans for each job is too large, it is difficult to model each job by employing the same method used in Experiment 2. Thus, to conduct this experiment, we need to re-model each job by the FTS based on the precedence relationships between operations of the experimental results of others. Experiments 6(a) and 6(b) are conducted based on experimental results of Wong et al. [65] and Zhang and Wong [70], respectively, and the results of GA-OP are also compared with them, respectively. The parameters for GA-OP are as follows: \( \text{PopSize} = 200, \ P_c = 0.8, \ P_m = 0.1, \) and \( \text{IteNo} = 200. \) In Table 6, the comparison of results is shown. As depicted in Table 6, GA-OP is effective for large-scale problems. Because GA-OP does not need to handle sequencing flexibility, the mean CPU time is substantially shorter than those of ACO and Enhanced ACO. Figure 14 shows the Gantt chart of Experiment 6(b).

**Table 6. Comparison of results on Experiment 6**

| Solution approach | Experiment 6(a) | Experiment 6(b) |
|-------------------|-----------------|-----------------|
|                   | ACO  | GA-OP | Enhanced ACO | GA-OP |
| Makespan          | 589  | 522   | 484           | 482   |
| Mean CPU time (ms)| 128,700 | 12,056 | 120534       | 12063 |

6.2. Performance analysis. Figure 15 illustrates the search capability and evolution speed of GA-OP in Experiment 4. It clearly shows that the curve decreases rapidly prior to the 50-th generation, and then the decrease rate slows down. At the 64-th generation, GA-OP obtains the same makespan as the final one of ALGA; at the 75-th generation, GA-OP obtains a better makespan than the final one of ALGA; and at the 95-th generation, GA-OP yields the final solution. Furthermore, GA-OP obtains a better makespan with only half of the population of ALGA [39]. Thus, GA-OP performs well in terms of convergence speed and solution quality.

6.3. Discussion. Overall, the experimental results indicate that the proposed approach is a more acceptable approach for the FJSP-OP. This can be attributed
mainly to the proposed encoding strategy. The advantages of the encoding strategy lie in: (1) we can select crossover/mutation operators independently for the three substrings in a chromosome, since these three substrings are independent; (2) almost any one of the classic crossover/mutation operators can be adopted for the OR substring and the machine assignment substring and can guarantee the legality of the offspring, since each gene in the OR substring or the machine assignment substring is randomly generated in the fixed interval \([1, \text{max} P]\) or \([1, \text{max} M]\); (3) the operation sequence substring inherits the characteristics of the operation-based representation.

On the other hand, the proposed encoding strategy has an underlying shortcoming that the coding spaces of the OR substring and the machine assignment substring are expanded and the mapping from chromosomes to solutions is an \(n\)-to-1 mapping. This is due to the fact that each gene in the OR substring or the machine assignment substring indirectly represents an OR-link path or a machine. It is reported that an EC-based algorithm with an encoding strategy of \(n\)-to-1 mapping may decrease computing efficiency [11]. However, this shortcoming is negligible when compared to the advantages of the proposed encoding strategy. This has been proven by the experimental results.
7. **Conclusions.** The FJSP-OP is a common and practical case of the IPPS problem that can be found in various application areas. A Four-Tuple Scheme is proposed for modelling and representing a job with operation and processing flexibility. It can easily create all possible process plans for a job by randomly generating different input arrays of the algorithm, GENERATE-PROCESS-PLAN. GA is employed as the underlying framework for solving the FJSP-OP and closely incorporated with the FTS. To improve the optimization performance, an efficient encoding and decoding strategy is developed. The use of this strategy ensures that the classic genetic operators can be directly used, and there is no need for extra repairing algorithms to guarantee the feasibility or legality of offspring in the evolutionary process. Experiments of different sizes are conducted to compare the proposed approach with other previous methods. The experimental results show that the proposed approach achieves significant improvements.

The encoding and decoding strategy proposed in this study is limited to the FJSP-OP. It cannot be directly employed to solve the standard IPPS problem. While considering and settling the problems caused by the sequencing flexibility in the encoding and decoding strategy, it can be expanded to solve the standard IPPS problem. This would be a fruitful area for further work.

**Acknowledgments.** The authors would like to thank the anonymous referees for their valuable comments that greatly helped to improve this paper. This study is supported by the National Science and Technology Plan of China under Grant No. 2015BAF09B01 and the China Scholarship under Grant No. 201606060170.

**Appendix1.** Experiment 3 is adopted from Shao et al. [58], and includes 6 jobs and 8 machines. The 6 jobs are shown in Figure A1.

![Figure A1. The data of Experiment 3](image-url)
Alternative machines of each operation performed for each job, processing times on the alternative machines of each operation, and the transportation times (the time units is same as processing time in Table A2) between the machines are given in Tables A1, A2 and A3, respectively.

The result of Experiment 3, shown in the Gantt chart of Figure 11, is given in terms of the starting and completion times of all operations as follow:

\[ M_1: (O_{46} : 0 - 22) (O_{56} : 39 - 61) (O_{16} : 61 - 71). \]
\[ M_2: (O_{11} : 0 - 18) (O_{51} : 18 - 36) (O_{63} : 41 - 93) (O_{39} : 93 - 115) (O_{49} : 115 - 137). \]
\[ M_3: (O_{31} : 0 - 12) (O_{62} : 25 - 37) (O_{69} : 97 - 119) (O_{19} : 119 - 145). \]
\[ M_4: (O_{61} : 0 - 22) (O_{21} : 22 - 44) (O_{59} : 118 - 145). \]
\[ M_5: (O_{15} : 23 - 46) (O_{33} : 46 - 75) (O_{17} : 75 - 107). \]
\[ M_6: (O_{32} : 17 - 40) (O_{57} : 66 - 90). \]
\[ M_7: (O_{47} : 30 - 74) (O_{28} : 74 - 118). \]
\[ M_8: (O_{24} : 47 - 67) (O_{58} : 97 - 115) (O_{29} : 121 - 145). \]

**Table A1.** Alternative machines of each operation performed for each job

| Operation | Job |
|-----------|-----|
| 1         | 2,4 | 2,4 | 3,5 | 3,5 | 2,4 | 2,4 |
| 2         | 7,8 | 7,8 | 4,6 | 4,6 | 3,5 | 3,5 |
| 3         | 1,2 | 1,2 | 2,3,5 | 2,3,5 | 1,2,6 | 1,2,6 |
| 4         | 8,6 | 8,6 | 6,7 | 6,7 | 6,8 | 6,8 |
| 5         | 3,5 | 3,5 | 1,8 | 1,8 | 1,7 | 1,7 |
| 6         | 1,2 | 1,2 | 1,4 | 1,4 | 1,3 | 1,3 |
| 7         | 5,6 | 5,6 | 6,7 | 6,7 | 6,7 | 6,7 |
| 8         | 1,7 | 1,7 | 5,8 | 5,8 | 5,8 | 5,8 |
| 9         | 3,8 | 3,8 | 4,2 | 4,2 | 4,3 | 4,3 |

**Table A2.** Processing times on the alternative machines of each operation

| Operation | Job |
|-----------|-----|
| 1         | 18,22 | 18,22 | 12,15 | 12,15 | 18,22 | 18,22 |
| 2         | 39,36 | 39,36 | 24,23 | 24,23 | 12,15 | 12,15 |
| 3         | 11,10 | 37,39 | 30,31,29 | 30,31,29 | 50,52,54 | 50,52,54 |
| 4         | 31,34 | 20,21 | 21,22 | 21,22 | 19,21 | 53,51 |
| 5         | 21,23 | 21,23 | 32,30 | 32,30 | 32,31 | 22,24 |
| 6         | 10,12,15 | 10,12,15 | 22,25 | 22,25 | 22,25 | 22,25 |
| 7         | 32,30 | 36,38 | 24,22 | 42,44 | 24,22 | 24,22 |
| 8         | 45,44 | 45,44 | 20,19 | 41,43 | 20,18 | 20,18 |
| 9         | 26,24 | 26,24 | 27,22 | 27,22 | 27,22 | 27,22 |

**REFERENCES**

[1] A. Azzouz, M. Ennigrou and L. B. Said, A hybrid algorithm for flexible job-shop scheduling problem with setup times, *Int. J. of Prod. Mgmt. & Eng.*, 5 (2017), 23–30.
Table A3. Transportation times between the machines

| Machine number | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  |
|----------------|----|----|----|----|----|----|----|----|
| 1              | 0  | 3  | 7  | 10 | 3  | 5  | 8  | 12 |
| 2              | 3  | 0  | 4  | 7  | 5  | 3  | 5  | 8  |
| 3              | 7  | 4  | 0  | 3  | 8  | 5  | 3  | 5  |
| 4              | 10 | 7  | 3  | 0  | 10 | 8  | 5  | 3  |
| 5              | 3  | 5  | 8  | 10 | 0  | 3  | 7  | 10 |
| 6              | 5  | 3  | 5  | 8  | 3  | 0  | 4  | 7  |
| 7              | 8  | 5  | 3  | 5  | 7  | 4  | 0  | 3  |
| 8              | 12 | 8  | 5  | 3  | 10 | 7  | 3  | 0  |

[2] K. R. Baker, Introduction to Sequencing and Scheduling, Wiley, New York, 1974.
[3] W. Banzhaf, The “molecula” traveling salesman, Bio. Cyber., 64 (1990), 7–14.
[4] A. Baykasoglu, Linguistic-based meta-heuristic optimization model for flexible job shop scheduling, Int. J. of Prod. Research, 40 (2002), 4523–4543.
[5] A. Baykasoglu, L. Ozbakir and A. Sonmez, Using multiple objective tabu search and grammars to model and solve multi-objective flexible job shop scheduling problems, J. of Intell. Manufacturing, 15 (2004), 777–785.
[6] S. Benjaafar and R. S. Ramakrishnan, Modelling, measurement and evaluation of sequencing flexibility in manufacturing systems, Int. J. of Prod. Research, 34 (1996), 1195–1220.
[7] C. Bierwirth, D. C. Mattfeld and H. Kopfer, On permutation representations for scheduling problems, in International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science, 1141, Springer, Berlin, 1996, 310–318.
[8] C. Bierwirth and D. Mattfeld, Production scheduling and rescheduling with genetic algorithms, Evol. Computation, 7 (1999), 1–17.
[9] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Annals of Ops. Research, 41 (1993), 157–183.
[10] I. A. Chaudhry and M. Usman, Integrated process planning and scheduling using genetic algorithms, Tehnicki Vjesnik, 24 (2017), 1401–1409.
[11] R. Cheng, M. Gen and Y. Tsujimura, A tutorial survey of job-shop scheduling problems using genetic algorithms— I. Representation, Comp. & Indust. Engineering, 30 (1996), 983–997.
[12] R. Cheng, M. Gen and Y. Tsujimura, A tutorial survey of job-shop scheduling problems using genetic algorithms: Part II. Hybrid genetic search strategies, Comp. & Indust. Engineering, 37 (1999), 51–55.
[13] G. Chryssolouris, S. Chan and W. Cobb, Decision making on the factory floor: An integrated approach to process planning and scheduling, Robotics & Comp. Integ. Manufacturing, 1 (1984), 315–319.
[14] G. Chryssolouris, S. Chan and N. P. Suh, An integrated approach to process planning and scheduling, CIRP Annals - Manufacturing Tech., 34 (1985), 413–417.
[15] D. A. Dabney, L. Green and V. Topalli, A priority-based heuristic algorithm (PBHA) for optimizing integrated process planning and scheduling problem, J. of Criminal Justice Edu., 2 (2015), 44–68.
[16] L. Davis, Job shop scheduling with genetic algorithms, In: International Conference on Genetic Algorithms, L. Erlbaum Associates Inc., Hillsdale, 1985, 136–140.
[17] H. H. Doh, J. M. Yu and J. S. Kim, D. H. Lee and S. H. Nam, A priority scheduling approach for flexible job shops with multiple process plans, Int. J. of Prod. Research, 51 (2013), 3748–3764.
[18] D. B. Fogel, An evolutionary approach to the traveling salesman problem, Bio. Cybernetics, 60 (1988), 139–144.
[19] J. Gao, M. Gen, L. Sun and X. Zhao, A hybrid of genetic algorithm and bottleneck shifting for multi-objective flexible job shop scheduling problems, Comp. & Indust. Engineering, 53 (2007), 149–162.
[20] K. Z. Gao, P. N. Suganthan, T. J. Chua, C. S. Chong, T. X. Cai and Q. K. Pan, A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion, Expert Syst. with Appl., 42 (2015), 7652–7663.
[21] M. Gen, Y. Tsujimura and E. Kubota, Solving job-shop scheduling problems by genetic algorithm, in IEEE International Conference on Systems, (1994), 576–579.
[22] M. Gen and R. Cheng, Genetic Algorithms and Engineering Optimization, John Wiley & Sons, 2000.
[23] D. E. Goldberg and R. Lingle, Alleles, loci and the traveling salesman problem, Proc. of 1st Int. Conf. on Genetic Algorithms and Their Applications, 12 (1985), 154–159.
[24] N. B. Ho, T. Yamakawa and K. M. Lee, A symbiotic evolution algorithm for the integration of process planning and job shop scheduling, European J. of Oper. Research, 179 (2007), 316–333.
[25] Y. K. Kim, K. Park and J. Ko, A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling, Comp. & Ops. Research, 30 (2003), 1151–1171.
[26] T. Kis, Job-shop scheduling with processing alternatives, European J. of Oper. Research, 151 (2003), 307–332.
[27] J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Mich., 1975.
[28] K. Ida and K. Oka, Flexible shop scheduling problem by genetic algorithm, Electrical Engineering in Japan, 177 (2011), 28–35.
[29] L. Jin, Q. Tang, C. Zhang, X. Shao and G. Tian, More MILP models for integrated process planning and scheduling, Int. J. of Prod. Research, 54 (2016), 4387–4402.
[30] B. Khoshhevis and Q. M. Chen, Integration of process planning and scheduling functions, J. of Intell. Manufacturing, 2 (1991), 165–175.
[31] Y. K. Kim, J. Y. Kim, and K. S. Shin, An asymmetric multileveled symbiotic evolutionary algorithm for integrated FMS scheduling, J. of Intell. Manufacturing, 18 (2007), 631–645.
[32] Y. K. Kim, K. Park and J. Ko, A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling, Comp. & Ops. Research, 30 (2003), 1151–1171.
[33] X. Y. Li, L. Gao, X. Shao, C. Zhang and C. Wang, Mathematical modeling and evolutionary algorithm-based approach for integrated process planning and scheduling, Comp. & Ops. Research, 37 (2010), 656–667.
[34] X. Y. Li, L. Gao and X. Y. Shao, An active learning genetic algorithm for integrated process planning and scheduling, Expert Syst. with Appl., 39 (2012), 6683–6691.
[35] X. Y. Li, X. Y. Shao, L. Gao and W. Qian, An effective hybrid genetic algorithm for integrated process planning and scheduling, Int. J. of Prod. Economics, 126 (2010), 289–298.
[36] X. Y. Li, C. Zhang, L. Gao, W. Li and X. Y. Shao, An agent-based approach for integrated process planning and scheduling, Expert Syst. with Appl., 37 (2010), 1256–1261.
[37] Y. J. Liu and J. J. Solberg, Effectiveness of flexible routing control, Int. J. of Flexible Manufacturing Syst., 3 (1991), 189–211.
[38] M. Liu, S. Yi and P. Wen, Quantum-inspired hybrid algorithm for integrated process planning and scheduling, Proceedings of the Inst. of Mech. Engineers Part B: J. of Engineering Manufacture, 232 (2016), 1–18.
[39] G. Luo, X. Wen, H. Li, W. Ming and G. Xie, An effective multi-objective genetic algorithm based on immune principle and external archive for multi-objective integrated process planning and scheduling, Int. J. of Adv. Manufacturing Tech., 91 (2017), 3145–3158.
[40] V. K. Manupati, G. D. Putnik, M. K. Tiwari, P. Ávila and M. M. Cruz-Cunha, Integration of process planning and scheduling using mobile-agent based approach in a networked manufacturing environment, Comp. & Industrial Engineering, 94 (2016), 63–73.
[41] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Second edition, Springer-Verlag, Berlin, 1994.
[47] C. Moon, Y. H. Lee, C. S. Jeong and Y. S. Yun, Integrated process planning and scheduling in a supply chain, Comp. & Indast. Engineering, 54 (2008), 1048–1061.

[48] N. Morad and A. Zalzala, Genetic algorithms in integrated process planning and scheduling, J. of Intell. Manufacturing, 10 (1999), 169–179.

[49] H. E. Nouri, O. B. Driss and K. Ghédira, Solving the flexible job shop problem by hybrid metaheuristics-based multiagent model, J. of Indust. Engineering Int., 14 (2018), 1–14.

[50] M. Nouriri, A. Bekrar, A. Jemai, S. Niar and A. C. Ammari, An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem, J. of Intell. Manufacturing, 29 (2018), 603–615.

[51] C. Özgüven, L. Özbakır and Y. Yavuz, Mathematical models for job-shop scheduling problems with routing and process plan flexibility, Appl. Math. Modelling, 34 (2010), 1539–1548.

[52] J. J. Palacios, M. A. González, C. R. Vela, I. González-Rodríguez and J. Puente, Genetic tabu search for the fuzzy flexible job shop problem, Comp. & Ops. Research, 54 (2015), 74–89.

[53] M. Petrovića, J. Petronijevića, M. Mitića, N. Vukovića, Z. Miljkovića and B. Babića, The ant lion optimization algorithm for integrated process planning and scheduling, Appl. Mechanics & Materials, 834 (2016), 187–192.

[54] R. K. Phanden, A. Jain and R. Verma, Integration of process planning and scheduling: A state-of-the-art review, Int. J. of Comp. Integrated Manufacturing, 24 (2011), 517–534.

[55] L. Qiao and S. Lv, An improved genetic algorithm for integrated process planning and scheduling, The Int. J. of Adv. Manufacturing Tech., 58 (2012), 727–740.

[56] A. Rajabinasab and S. Mansour, Dynamic flexible job shop scheduling with alternative process plans: An agent-based approach, The Int. J. of Adv. Manufacturing Tech., 54 (2011), 1091–1107.

[57] C. Saygin and S. E. Kilic, Integrating flexible process plans with scheduling in flexible manufacturing systems, The Int. J. of Adv. Manufacturing Tech., 15 (1999), 268–280.

[58] X. Y. Shao, X. Y. Li, L. Gao and C. Y. Zhang, Integration of process planning and scheduling—a modified genetic algorithm-based approach, Comp. & Ops. Research, 36 (2009), 2082–2096.

[59] O. Sobeyko and L. Mi¶Nch, Integrated process planning and scheduling for large-scale flexible job shops using metaheuristics, Int. J. of Prod. Research, 55 (2017), 392–409.

[60] D. Sreeramulu, Y. Sagar, P. Suman and A. S. Kumar, Integration of process planning and scheduling of a manufacturing systems using petri nets and genetic algorithm, Indian J. of Science & Tech., 9 (2016), 2–8.

[61] G. Syswerda, Uniform Crossover in Genetic Algorithms, International Conference on Genetic Algorithms, (1989), 2–9.

[62] Z. Wang and T. Ju, The research about integration of process planning and production scheduling based on genetic algorithm, International Conference on Computer Science and Software Engineering, (2008), 9–12.

[63] T. N. Wong, C. W. Leung, K. L. Mak and R. Y. K. Fung, An agent-based negotiation approach to integrate process planning and scheduling, Int. J. of Prod. Research, 44 (2006), 1331–1351.

[64] T. N. Wong, C. W. Leung, K. L. Mak and R. Y. K. Fung, Integrated process planning and scheduling/rescheduling—an agent-based approach, Int. J. of Prod. Research, 44 (2006), 3627–3655.

[65] T. N. Wong, S. C. Zhang, G. Wang and L. P. Zhang, Integrated process planning and scheduling – multi-agent system with two-stage ant colony optimisation algorithm, Int. J. of Prod. Research, 50 (2012), 6188–6201.

[66] H. Xia, X. Li and L. Gao, A hybrid genetic algorithm with variable neighborhood search for dynamic integrated process planning and scheduling, Comp. & Indast. Engineering, 102 (2016), 99–112.

[67] L. Yin, L. Gao, X. Li and H. Xia, An improved genetic algorithm with rolling window technology for dynamic integrated process planning and scheduling problem, IEEE International Conference on Computer Supported Cooperative Work in Design, (2017), 414–419.

[68] Y. Yuan and H. Xu, Multiobjective flexible job shop scheduling using memetic algorithms, IEEE Transactions on Automation Science and Engineering, 12 (2015), 336–353.

[69] G. Zhang, L. Zhang, X. Song, Y. Wang and C. Zhou, A variable neighborhood search based genetic algorithm for flexible job shop scheduling problem, Cluster Computing, (2018), 1–12.

[70] S. Zhang and T. N. Wong, Integrated process planning and scheduling: An enhanced ant colony optimization heuristic with parameter tuning, J. of Intell. Manufacturing, 29 (2018), 585–601.
[71] L. Zhang and T. N. Wong, *An object-coding genetic algorithm for integrated process planning and scheduling*, *European J. of Oper. Research*, 244 (2015), 434–444.

[72] S. Zhang, Z. Yu, W. Zhang, D. Yu and Y. Xu, *An extended genetic algorithm for distributed integration of fuzzy process planning and scheduling*, *Math. Problems in Engineering*, 3 (2016), 1–13.

Received November 2017; 1st revision October 2018; 2nd revision March 2019.

E-mail address: huangxuewen@dlut.edu.cn
E-mail address: zhangxiaotongdlut@163.com
E-mail address: Sardar.Islam@vu.edu.au
E-mail address: carlosveme@unisabana.edu.co