IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 2, 2020, accepted August 25, 2020, date of publication September 8, 2020, date of current version September 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3022655

A Hybrid Estimation-of-Distribution Algorithm for Scheduling Flexible Job Shop With Limited Buffers Based on Petri Nets

ZHENXIN GAO, YANXIANG FENG, AND KEYI XING
State Key Laboratory for Manufacturing Systems Engineering, Xi'an Jiaotong University, Xi'an 710049, China
Systems Engineering Institute, Xi'an Jiaotong University, Xi'an 710049, China

Corresponding authors: Yanxiang Feng (fengyxss@stu.xjtu.edu.cn) and Keyi Xing (kyxing@xjtu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61573278 and in part by the National Science Foundation for Post-Doctoral Scientists of China under Grant 2018M643660.

ABSTRACT This article focuses on the production scheduling problem in the flexible job shop (FJS) environment with limited buffers. Limited manufacturing resources and buffers may lead to blockage and deadlock phenomenon. In order to establish production scheduling with minimum makespan, the timed Petri net (PN) model of a production process is established. Based on this PN model, a novel Hybrid Estimation-of-Distribution Algorithm (HEDA) is proposed for solving the considered scheduling problem. A candidate solution for the problem is coded as an individual that consists of a route sequence for processing jobs and a permutation with repetition of jobs. A deadlock prevention policy is used to check the feasibility of individuals, such that it can be decoded into a feasible sequence of transitions, i.e., a feasible schedule. By using an effective voting procedure of elite individuals, two probability models in HEDA corresponding to different subsections of individuals are constructed. Based on the probability models, offspring individuals are then produced. As an improvement strategy, simulated-annealing-based local search is designed and incorporated into HEDA to enhance the entire algorithm's search ability. The proposed hybrid HEDA is tested on FJS examples. The results show its feasibility and effectiveness.

INDEX TERMS Flexible job shop (FJS), limited buffers, scheduling, hybrid estimation-of-distribution algorithm (HEDA), petri net (PN).

I. INTRODUCTION

In the classical scheduling problems, whether job-shop or flow-shop scheduling problems (JSSP or FSSP), infinite sizes of buffers for storing jobs are usually assumed [1], [2]. While, in many real production systems, the buffer space for storing jobs is usually limited, such as in flow shop [3]–[8], job-shop [1], [9]–[19], and automated manufacturing systems [20]. In these manufacturing systems, the total resources (machines, transportation equipment, buffers, etc.) to hold the jobs are limited, and hence in the process of system operation, if the resources are not allocated properly, it often encounters blocking and/or even deadlock [11]–[13], [16], [20]–[24], [42]–[44]. Therefore, in order to run such manufacturing systems effectively, we need to consider two problems: liveness control and optimal scheduling. The goal of control is to ensure the normal operation of a system without deadlock, so as to complete all production tasks, while scheduling is to allocate resources to tasks reasonably, so as to optimize the (some) performance of the system on the basis of liveness control. These problems create a relatively new and more significant research direction, and have attracted many researchers [11], [12], [20]–[25].

Papadimitriou and Kanellakis [6] studied the complexity of FSSP with limited buffers and shown that even the simplest two machine flow-shop problem with a limited buffer between two machines is strongly NP-hard. Hence, to solve scheduling problems of manufacturing system with limited buffer capacities, especially, deadlock-prone manufacturing system, in reasonable time, heuristics and metaheuristics have to be applied. So far, many approximate algorithms have been developed in the literature for solving such complex manufacturing system scheduling problems.

On FSSP with limited buffers, many studies have been done and most works concern the problem with makespan objective [4], [5], [7], [8], [26], [27]. Compared to the

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaoou Li.

FSSP with limited buffer constraints in the literature, JSSP with limited buffer has received much less attention [9], [11]–[14], [19], and in these existing studies, it is focused on the special cases, where all buffers have capacity 0, called the blocking job-shop problem [28]–[31]. In the literature, one of the main reasons for neglecting the impact of finite buffers is that compared with flexible job shop (FJS) scheduling problem, FJS scheduling problem with finite buffer, although both are NP-hard, is more complex and contains more constraints; while the other and most important reason is that the limited capacity buffer constraints in the job-shop environment leads to so-called deadlocks, and the detection and resolution of such deadlocks is also NP-hard. Mascis and Pacciarelli [30] studied JSSP with blocking and no-wait constraints, and established the complexity results of the problem. Brucker *et al.* [9] and Heitmann [14] investigated several types of JSSPs with limited buffer constraints by classifying buffers into three classes: (i) machine-dependent output buffers, (ii) machine-dependent input buffers, and (iii) job-dependent buffers. Based on the alternative and disjunctive graph models, representations of feasible solutions for job shop with limited buffers are investigated in [9] and a constructive heuristic to find feasible solutions is presented. Fahmy *et al.* [11], [12] presented an insertion heuristic based on matrices and Latin rectangles. This heuristic is capable to take into account limited buffer capacities and to avoid deadlocks, and hence, can obtain a feasible schedule. However, the computation time is longer, especially when applying the procedure within metaheuristics. Pranzo and Pacciarelli [16] proposed an iterative greedy algorithm to solve two types of blocking job shop scheduling problems, one with swap allowed and the other with no swap allowed. The need to swap jobs between machines (and buffers) arises whenever there is a circular set of blocking or deadlock jobs in which each job is waiting for a machine (or a buffer space) occupied by other jobs in the same set, that is, this set of jobs are in deadlock state. By swapping jobs, deadlock is resolved. But this swapping actually requires additional equipment to complete. For JSSP with buffer constraints and jobs consuming variable buffer space, Witt and Voβ [19] presented a heuristic to find feasible solutions. Gomes *et al.* [13] investigated the scheduling problem of flexible job shop (FJS) with groups of parallel homogeneous machines and limited intermediate buffers, discrete parts manufacturing industries that operate on a make-to-order basis. Under the assumption that after the job is processed on the machine, there must be buffer space to store it, the integer linear programming model of the scheduling problem is developed, and by solving this integer linear programming, an optimal schedule is obtained.

Automated manufacturing systems, especially flexible manufacturing systems (FMS) can be considered as a generalization of the FJSs with limited capacity buffers. Both of them are faced with deadlock and have the same deadlock characteristics. The deadlock problem of FMSs has been widely studied, and many deadlock control policies are presented [20]–[24]. Deadlocks in FJSs with limited capacity buffers can be handled according to deadlock control methods in FMSs. These methods provide a necessary and feasible control basis for solving the scheduling problem of FJSs with limited buffers.

This article focuses on the production scheduling problem of flexible job shop manufacturing environment with limited capacity buffers, where buffers are machine-dependent, that is, a machine and its buffer form a manufacturing cell or workstation, and the buffer is used to store the jobs that need to be processed or have been finished on the machine. This kind of buffers is widely used in automatic production line [9], [12], [14], [18], [28], railway network [10], [29], aircraft traffic control [17], and so on. The shop manufactures medium-volume discrete jobs (or, parts, products) of different types in a make-to-order basis as in Gomes *et al.* [13]. An order or a type of jobs, consists of a number of jobs to be produced and their processing routes. The re-circulation of jobs in the considered flexible job shops is permitted, that is, jobs can visit some machines more than once, as in FMSs. Furthermore, as in the classical job-shop problem, we assume that all jobs are available at the beginning time, and each job leaves the system directly after the finishing of its last operation, i.e. that sufficient buffer space is available to store all completed jobs. For such a manufacturing shop, the scheduling problem with the completion time as the optimization goal is investigated in this study. From the above literature review, the scheduling problem of such FJS with limited capacity buffers is rarely studied. In fact, to our best knowledge, no work has provided a systematic study and a feasible solution to this scheduling problem.

Note that FJS scheduling problem (with infinite buffers) and FJS scheduling problem with limited buffers have essential differences. To solve flow shop and flexible job shop scheduling problems, many algorithms have been proposed, such as particle swarm optimization [4], discrete differential evolution algorithm [5], artificial bee colony algorithm [27], estimation of distribution algorithm [33], and genetic algorithm [35]. In these algorithms, the individual's feasibility is determined by the individual's encoding, that is to say, each candidate individual can be decoded into a feasible schedule. For FJSs with limited buffers, the feasibility of an individual cannot be guaranteed by encoding, that is to say, a candidate individual may not be able to decode into a feasible schedule. Therefore, it is necessary to detect the feasibility of individuals and correct the infeasible individuals into the feasible individuals. From this point, we can know that the FJS scheduling problem with infinite buffers is totally different from that with limited buffers, and the evolutionary algorithms for the former has no deadlock avoidance mechanism, and hence, cannot be directly used for the latter.

This article uses place-timed Petri nets (PNs) to model FJSs with limited buffers. Petri net is an effective tool for modeling discrete event systems [37]–[41]. In view of the complexity of the considered scheduling problem, the estimation of distribution algorithm (EDA) is used to solve it. EDA is an evolutionary algorithm that has received much attentions of many researchers [32]. It uses neither crossover nor mutation operator, but reproduces offsprings based on a probabilistic model learned from a population of parents. This model-based approach to optimization allows EDA to successively solve many complex and large problems [32], [33]. In this article, a candidate solution for the scheduling problem is denoted as an individual. Because of route flexibility, a feasible solution not only specifies a processing route for each job, but also determines a processing sequence of operations of all jobs. Hence, an individual designed in this article consists of two parts. The first part is a sequence of routes for jobs and the second one is a permutation with repetition of jobs. By letting the i-th occurrence of a job in the permutation of an individual correspond to the i-th operation of the job and using route information in the first part, the individual can be decoded as a candidate solution. Note that such a candidate solution may be not feasible, and can cause deadlock. In this article, by embedding a deadlock detection and avoidance policy into a decoding process, the feasibility of a candidate solution obtained by decoding an individual can be guaranteed.

Corresponding to the structure of individuals, two probabilistic models, route and operation probabilistic models, are generated by a simple vote of elite individuals and taking into account their weights. Based on these probabilistic models, offspring individuals are then produced. Moreover, to balance the global and local searches and to further improve the performance of EDA, the simulated annealing based local search is designed and incorporated into EDA. The proposed hybrid EDA is tested on a set of FJS examples, showing its feasibility and effectiveness.

The rest of this article is organized as follows. Section II introduces the considered FJSs with limited buffers and batch production, and develops their PN models. A novel hybrid estimation of distribution algorithm for the considered FJS is described in Section III. An example is given to show the effectiveness of the proposed scheduling method in Section IV. Section V concludes this article.

II. PRELIMINARY

In this section, we first introduce the considered FJSs and then establish their Petri net models. For concepts and notations of Petri nets, a reader is referred to [34].

A. BASIC DEFINITIONS AND NOTATIONS OF PNS

A PN is a three-tuple $N = (P, T, F)$, where P and T are are finite sets of places and transitions, respectively., respectively, $F \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs. Given a node $x \in P \cup T$, the preset and post-set of x are defined as ${}^\bullet x = \{y \in P \cup T | (y, x) \in F\}$ and $x^\bullet = \{y \in P \cup T | (x, y) \in F\}$, respectively. These notations can be extended to a set, for example, let $S \subseteq P \cup T$, then ${}^\bullet S = \cup^\bullet_{x \in S} x$ and $S^\bullet = \cup_{x \in S} x^\bullet$.

A marking of N is a mapping $M : P \to Z$ where $Z \equiv \{0, 1, 2, \ldots\}$. Given a place $p \in P$ and a marking M, $M(p)$ denotes the number of tokens in p at M. Let $S \subseteq P$ be a set of places; the total number of tokens in all places of S at M is denoted by $M(S)$, i.e., $M(S) = \sum_{p \in S} M(p)$. A PN N with an initial marking M_0 is called a marked PN, denoted as (N, M_0).

A transition $t \in T$ is enabled at marking M, denoted by $M[t >$, if $\forall p \in {}^\bullet t$, $M(p) \geq 1$. An enabled transition t at M can be fired, resulting in a new marking M_1, denoted by $M[t > M_1$, where $M_1(p) = M(p) - 1$, $\forall p \in {}^\bullet t \setminus t^\bullet$, $M_1(p) = M(p) + 1$, $\forall p \in t^\bullet \setminus {}^\bullet t$, and, otherwise, $M_1(p) = M(p)$.

A sequence of transitions $\alpha = t_1 t_2 \ldots t_k$ is *feasible* from marking M if there exists $M_i[t_i > M_{i+1}$, $\forall i \in Z_k \equiv \{1, 2, \ldots, k\}$, where $M_1 = M$. We state that M_i is a reachable marking from M. Let $R(N, M_0)$ denote the set of all reachable markings of N from M_0.

A path is a *string* $\tau = x_1 x_2 \ldots x_k$, where $x_i \in P \cup T$ and $(x_i, x_{i+1}) \in F$, $\forall i \in Z_{k-1}$. A path $\tau = x_1 x_2 \ldots x_k$ is a *circuit* if $x_1 = x_k$.

B. FJS WITH LIMITED BUFFERS

The FJS considered in this article consists of u workstations, $w_1 - w_u$, and can process v types of jobs, $q_1 - q_v$. A workstation is a machine with a finite buffer. The machine is used to process jobs, while the buffer is for staging and storing jobs.

Let $W = \{w_i, i \in Z_u\}$ and $Q = \{q_i, i \in Z_v\}$. Suppose that workstation w_i has machine m_i and its buffer capacity is $C(w_i)$, i.e., it can simultaneously hold at most $C(w_i)$ jobs. Each job only occupies a unit buffer space at any time and the machine is not idle as long as there are unprocessed jobs in its buffer.

The considered FJS supports batch processing and route flexibility. That is, there are multiple jobs of the same type to be processed, and a job may have multiple processing routes. A processing route of a job is an ordered sequence of operations to be processed on machines with specified processing time. The same type of jobs has the same set of processing routes. Let $\varphi(q)$ be the number of type-q jobs to be processed, and $n = \sum_{q \in Q} \varphi(q)$, the total number of jobs.

Suppose that type-q jobs have $\mu(q)$ processing routes $\pi_1 - \pi_{\mu(q)}$. Route π_i can be expressed as $\pi_i = o_{i1} o_{i2} \ldots o_{il_i}$, where o_{ij} is the j-th operation in π_i and $l_i \equiv \lambda(\pi_i)$ is the length of route π_i. In this article, suppose that each operation requires only one predetermined machine for processing, and the processing time of operation o_{ij} is $d(o_{ij})$. Therefore, a processing route corresponds to a sequence of machine or workstations. Let $w(o_{ij})$ denote the workstation with the machine required for processing operation o_{ij}. Then, π_i is determined by the sequence of workstations $w(\pi_i) = w(o_{i1}) w(o_{i2}) \ldots w(o_{il_i})$. Let $\chi(q) = \max\{\lambda(\pi_i) | \pi_i$ is a processing route for type-q jobs$\}$. Since the same type of jobs has the same set of processing routes, we also use $\chi(J)$ to denote the maximum length of processing routes for job J, that is, if J is a type-q job, then $\chi(J) = \chi(q)$.

For convenience's sake, to type-q jobs, two fictitious operations, o_{qs} and o_{qe}, are added, while $w(o_{qs}) \equiv b_{qs}$ and $w(o_{qe}) \equiv b_{qe}$ are two fictional infinite buffers used to store raw and processed type-q jobs, respectively. Then, route π_i is extended and still denoted as π_i, i.e., $\pi_i = o_{qs} o_{i1} o_{i2} \ldots o_{il_i} o_{qe}$, or $w(\pi_i) = w(o_{qs}) w(o_{i1}) w(o_{i2}) \ldots w(o_{il_i}) w(o_{qe})$.

To perform its operation o_{ij}, a type-q job J_i first enters the buffer of workstation $w(o_{ij})$, and then, when the machine in $w(o_{ij})$ is available, it is processed for $d(o_{ij})$ time units without preemption. During the time in workstations, no matter what state a job is in (waiting to be processed, being processed, or has been processed), it always occupies a unit buffer space.

The scheduling objective is to minimize the completion time of the last job to leave the system or makespan.

C. PN MODEL OF FJS WITH LIMITED BUFFERS

In this article, Petri nets are used to model the considered FJSs. To establish the PN model of such an FJS with limited buffers, we first model processing routes of jobs, and then the request, utilization, and release of resources (machine and buffers) by jobs in workstations.

For a type-q job J, one of its routes, $\pi_i = o_{qs}o_{i1}o_{i2}\ldots o_{il}$ o_{qe}, is modeled by a path of transitions and operation places, denoted as $O(\pi_i) = p_{qs}t_{i1}p_{i11}t_{i11}p_{i12}t_{i12}p_{i13}t_{i2}p_{i21}t_{i21}p_{i22}$ $t_{i22}p_{i23}t_{i3}\ldots t_{il}p_{il1}t_{il1}p_{il2}t_{il2}p_{il3}t_{i(l+1)}p_{qe}$, and called as an operation path (O-path), where operation places p_{qs} and p_{qe} represent fictitious operations o_{qs} and o_{qe}, respectively. Operation o_{ij} of job J is processed in workstation $w(o_{ij})$. Its activities in $w(o_{ij})$ are simulated by path $t_{ij}p_{ij1}t_{ij1}p_{ij2}t_{ij2}p_{ij3}t_{i(j+1)}$. The firing of transition t_{ij} implies that job J leaves the current workstation $w(o_{i(j-1)})$ or b_{qs} and enters the buffer of the next workstation $w(o_{ij})$ or b_{qe} if $o_{i(j-1)}$ is the last operation of the job. To make it clear that the relationship between transition t_{kl} and operation o_{ij}, the notation $t_{ij}[o_{ij}]$ will be used, and t_{kl} is called the *preparatory transition* of operation o_{ij}. The firings of transitions t_{ij1} and t_{ij2} represent the processing beginning and end of operation o_{ij}, respectively. Places p_{ij1} and p_{ij3} are used to store the jobs whose operation o_{ij} has not started and has completed, respectively, so they are non-timed, that is, the sojourn time of tokens in them is 0 and can leave at any time. Place p_{ij2} represents that operation o_{ij} is being processed by machine, and hence it is timed, and the sojourn time of the token in place p_{ij2} is the processing time of operation o_{ij}, that is, the sojourn time is $d(o_{ij})$.

Hence, the PN model of routes for type-q jobs can be denoted as

$$(N_q, M_{q0}) = (P_{Oq} \cup \{p_{qs}, p_{qe}\}, T_q, F_q, M_{q0})$$

where P_{Oq} is the set of operation places, p_{ij1}, p_{ij2}, p_{ij3}, corresponding to various states of jobs in the workstations. T_q and F_q are the sets of all transitions and arcs in all O-paths for type-q jobs. M_{q0} is the initial marking, $M_{q0}(p_{qs}) = \varphi(q)$ and $M_{q0}(p) = 0$, $\forall p \in P_{Oq} \cup \{p_{qe}\}$.

In N_q, $\forall t \in T_q$, $|{}^{\bullet}t| = |t^{\bullet}| = 1$, that is, N_q is a state machine, and consists of all O-paths from p_{qs} to p_{qe}. Such a path corresponds to a processing route of type-q jobs. A place $p \in P_q$ is called a split place if $|p^{\bullet}| > 1$. From a split place, jobs can choose their future processing routes.

In order to model the request and release of resources (buffers or machines) in Petri net, assign two places corresponding to workstation w_k and its machine m_k respectively, denoted also by w_k and m_k for simplicity.

A token in w_k represents an available unit buffer space. The initial marking of w_k is $C(w_k)$. Let $P_W = \{w_1, \ldots, w_u\}$, the set of all workstation places. Similarly, let $P_M = \{m_1, \ldots, m_u\}$ the set of all machine places.

A token in m_k indicates that machine m_k is idle and available. Since we assume that buffers are machine-dependent in this article, there is only one machine per workstation. That is, the initial marking of place m_k is 1.

Now consider the request and release of resources. Let $O(\pi_i) = p_{qs}t_{i1}p_{i11}t_{i11}p_{i12}t_{i12}p_{i13}t_{i2}p_{i21}t_{i21}p_{i22}t_{i22}p_{i23}t_{i3}\ldots$ $t_{il}p_{il1}t_{il1}p_{il2}t_{il2}p_{il3}t_{i(l+1)}p_{qe}$ be an O-path of type-q jobs. If the operation corresponding to p_{ij2} is processed by machine m_k in workstation w_k, then add arcs (w_k, t_{ij}) and $(t_{i(j+1)}, w_k)$, representing the job entering and leaving w_k respectively. At the same time, arcs (m_k, t_{ij1}) and (t_{ij2}, m_k) are added to simulate the start and end of the operation processing on m_k respectively.

Let F_W denote the set of all arcs related with workstation and machine places. Then, the activities of all jobs among workstations can be modeled by the following Petri net, called as PN for scheduling (PNS).

$$(N, M_0) = (P_O \cup P_S \cup P_E \cup P_W \cup P_M, T, F, M_0)$$

where $P_O = \cup_{q\in Q}P_{Oq}$, $P_S = \{p_{qs}|q \in Q\}$, $P_E = \{p_{qe}|q \in Q\}$, $T = \cup_{q\in Q}T_q$, $F = F_Q \cup F_W$, and $F_Q = \cup_{q\in Q}F_q$. The initial marking M_0 is defined as $M_0(p_{qs}) = \varphi(q)$, $\forall p_{qs} \in P_S$, $M_0(p) = 0$, $\forall p \in P_O \cup P_E$, and $M_0(w) = C(w)$, $\forall w \in P_W$.

Let us use the following example to illustrate the modeling method.

Example 1: Consider an FJS with five workstations $w_1 - w_5$. The buffer capacities of workstations are 1, 1, 1, 2, and 2, respectively, i.e., $C(w_1) = C(w_2) = C(w_3) = 1$ and $C(w_4) = C(w_5) = 2$. The system can process two job types, types-A and B, with 3 and 2 jobs to be processed, respectively, i.e., $\varphi(A) = 3$ and $\varphi(B) = 2$. Type-A jobs can be processed through $w_1w_2w_3$ or $w_1w_4w_5w_3$, while type-B jobs through $w_3w_5w_1$. Then the PN model of jobs through workstations is shown in Figure 1, where three processing routes π_1, π_2, and π_3 are modeled by O−paths $\mathrm{O}(\pi_1) = p_{As}t_{11}p_{111}t_{111}p_{112}t_{112}p_{113}t_{12}p_{121}t_{121}p_{122}$ $t_{122}p_{123}t_{13}p_{131}t_{131}p_{132}t_{132}p_{133}t_{14}p_{Ae}$, $\mathrm{O}(\pi_2) = p_{As}t_{11}p_{111}$ t_{111} $p_{112}t_{112}$ $p_{113}t_{22}p_{221}t_{221}p_{222}$ $t_{222}p_{223}t_{23}p_{231}t_{231}p_{232}t_{232}$ $p_{233}t_{24}p_{131}t_{131}p_{132}t_{132}p_{133}t_{14}p_{Ae}$, and $\mathrm{O}(\pi_3) = p_{Bs}$ $t_{31}p_{311}t_{311}p_{312}$ $t_{312}p_{313}t_{32}p_{321}t_{321}p_{322}t_{322}p_{323}t_{33}p_{331}t_{331}p_{332}$ $t_{332}p_{333}t_{34}p_{Be}$, respectively.

When all the jobs have been processed, the system reaches the final state where no job is in the system and all resources are available or idle, or (N, M_0) reaches the final marking M_f, where $M_f(p) = 0$, $\forall p \in P_O \cup P_S$; $M_f(p) = M_0(p)$, $\forall p \in P_W \cup P_M$; $M_f(p_{qe}) = M_0(p_{qs})$, $\forall p_{qe} \in P_E$.

Let α be a sequence of transitions of (N, M_0). If α can lead (N, M_0) from M_0 to M_f, i.e., $M_0[\alpha > M_f$, it is called to be feasible. Under the assumption that all transitions on α are fired as early as possible, α is a (feasible) schedule of the system, and in this case, the firing time of the last transition in α is the makespan of schedule α. The scheduling problem considered

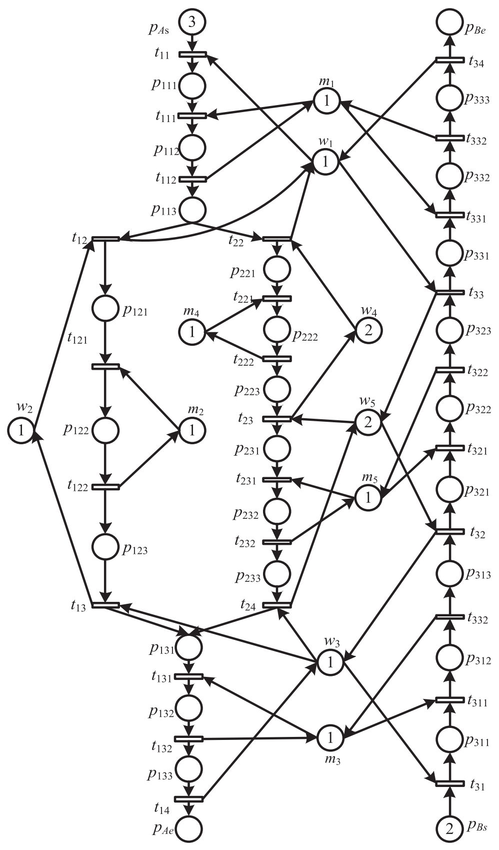


FIGURE 1. The PNS model of an FJS.

in this article is to find one with minimum makespan among all feasible sequences.

III. PROPOSED ALGORITHM

Papadimitriou and Kanellakis have proved in [6] that the two machines FSSP with limited buffers, as a particular case of our problem, is strongly NP-hard. So when the problem size increases, it becomes impractical to obtain the optimum solution of the considered scheduling problem within a reasonable time. Thus intelligent optimization methods [35] are widely used. In this article, we introduce a new HEDA to solve our scheduling problem. The proposed HEDA is the combination of basic EDA, local search, and DAPs. Its main components are as follows.

- Representation and amending of individuals;
- Fitness function;
- Initialization;
- Probabilistic model and generating new individuals;
- Local search.

In the rest part of the paper, we use $J = (J_1, J_2, \ldots, J_n)$ to denote a permutation with a given order of all jobs, and suppose that there are a total of K job processing routes. Then $\pi = (\pi_1, \pi_2, \ldots, \pi_K)$ is used to denote a route permutation with a fixed order. For example, in Example 1, there are five jobs to be processed through three routes $\pi_1 - \pi_3$, three type-A jobs, denoted as $J_1 - J_3$, and two type-B jobs, J_4 and J_5. Then $J = (J_1, J_2, J_3, J_4, J_5)$ and $\pi = (\pi_1, \pi_2, \pi_3)$ are given job and route permutations, respectively.

A. REPRESENTATION AND AMENDING OF INDIVIDUALS

1) INDIVIDUAL REPRESENTATION

In our HEDA, a permutation with repetition of jobs is used to represent operation information of jobs and as a part of individual coding. On the other hand, a job may have different processing routes, and hence, the route information of jobs is also included in individual coding. Because the route of a job is unique within a workstation, in order to simplify the individual coding, the coding used in this article is only limited to the operation level or the corresponding workstation level, and the activities of jobs in workstations will be arranged according to the principle of first arrival and first processing. That is, an individual I contains two parts: operation part S_o and route part S_r, i.e., $I = (S_o; S_r)$, where each type-q job appears $\chi(q)$ times in S_o, and $S_r = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is an n-dimension vector and σ_k is a processing route of job J_k specified by I. Note that elements of S_r and J form one-to-one correspondence relation.

For a given individual $I = (S_o; S_r)$, let the i-th appearance of type-q job J_k in S_o represent the i-th operation of J_k in route π_k, omitting redundant J_k if the length of π_k is less than $\chi(q)$, i.e., $\lambda(\pi_k) < \chi(q)$. In such a way, S_o is translated into a sequence of operations, denoted as $\Delta(S_o)$. Then, according to the given route for each job in S_r, and matching each operation with its preparatory transition in (N, M_0), $\Delta(S_o)$ or I is interpreted as a sequence of transitions in Petri net model, denoted as $\alpha'(I)$. Thus, any individual can be decoded into a sequence of transitions in (N, M_0).

For a given individual $I = (S_o; S_r)$, although $\Delta(S_o)$ contains all operations of jobs to be processed, $\alpha'(I)$ is not a complete sequence of firing transitions from M_0 to M_f. In $\alpha'(I)$, there is a lack of transitions that represent the activities of jobs in workstations, as well as the last transition of the processing route to each job.

Note that the activity sequence or route of jobs in each workstation is uniquely determined, taking the form of $t_{ij}p_{ij1}t_{ij1}p_{ij2}t_{ij2}p_{ij3}$, and as long as a job enters a workstation, its corresponding operation can always be completed, that is, jobs in p_{ij1} and p_{ij2} always reach p_{ij3} as time goes on. This kind of token transfer only takes up the processing time of the machine, and has no effect on the liveness of the system. Therefore, it can be considered that the token generated by t_{ij}-firing directly reaches p_{ij3}, and for simplicity, transitions t_{ij1} and t_{ij2} are omitted in the transition sequence of coding $\alpha'(I)$.

On the other hand, the processing order of jobs in the same workstation can be arranged in many ways, such as in first come first processing, or the same type of jobs can be put together for continuous processing as much as possible if the set-up time of machines is considered. Therefore, in order to reduce the encoding length, this article does not code the activity order of jobs in the workstations, but leaves it in the simulation algorithm, and the principle of "first come first process" is adopted in the simulation of the proposed algorithm.

Therefore, in order to make $\alpha'(I)$ complete, we only need to add the last transition of the processing route of each job to the back of $\alpha'(I)$, and denote the resulting transition sequence as $\alpha(I)$.

Example 2: Consider the FJS in Example 1. Its PNS model is shown in Figure 1. There are five jobs to be processed: three type-A jobs, denoted as $J_1 - J_3$, and two type-B jobs, denoted as J_4 and J_5. Type-A jobs have two processing routes π_1 and π_2, while type-B jobs have only one route π_3. Then, $S_{r1} = (\pi_2, \pi_2, \pi_1, \pi_3, \pi_3)$ is the route part of an individual, where the routes of J_1, J_2, J_3, J_4, and J_5 are set to π_2, π_2, π_1, π_3, and π_3, respectively. Since $\lambda(\pi_1) = 3$, $\lambda(\pi_2) = 4$, $\lambda(\pi_3) = 3$, we know that $\chi(A) = \max\{\lambda(\pi_1), \lambda(\pi_2)\} = 4$, and $\chi(B) = \lambda(\pi_3) = 3$. Then the numbers of type-A and B jobs that are contained in the operation section of an individual should be 4 and 3, respectively. For example, $S_{o1} = (J_2, J_2, J_2, J_1, J_4, J_1, J_1, J_3, J_3, J_4, J_5, J_1, J_5, J_4, J_2, J_3, J_5, J_3)$ can be regarded as the operation part of an individual. Then, $I_1 = (S_{o1}; S_{r1}) = (J_2, J_2, J_2, J_1, J_4, J_1, J_1, J_3, J_3, J_4, J_5, J_1, J_5, J_4, J_2, J_3, J_5, J_3; \pi_2, \pi_2, \pi_1, \pi_3, \pi_3)$ represents an individual. According to the given route in S_r, we can obtain the sequence of operations corresponding to S_{o1}, $\Delta(S_{o1}) = (o_{21}, o_{22}, o_{23}, o_{11}, o_{41}, o_{12}, o_{13}, o_{31}, o_{32}, o_{42}, o_{51}, o_{14}, o_{52}, o_{43}, o_{24}, o_{33}, o_{53})$, where o_{ki} is the i-th operation of job J_k. Then by matching an operation with its preparatory transition, we have the sequence of transitions corresponding to I_1, $\alpha'(I_1) = (t_{11}, t_{22}, t_{23}, t_{11}, t_{31}, t_{22}, t_{23}, t_{11}, t_{12}, t_{32}, t_{31}, t_{24}, t_{32}, t_{33}, t_{24}, t_{13}, t_{33})$, where commas are added just for clarity.

Note that $\chi(A) = 4$, and each of J_1, J_2, and J_3 appears 4 times in S_{o1}. The route for J_3 specified in S_{r1} is π_1 and has 3 operations, and hence, the 4-th J_3 in S_{o1} is redundant in converting S_{o1} to $\Delta(S_{o1})$. On the other hand, the second operation of J_3, o_{32}, is processed in w_2, and its preparatory transition is t_{12}, that is we have $t_{12}[o_{32}]$; while the routes for J_1 and J_2 given by S_{r1} are π_2. Then their second operations o_{12} and o_{22} are processed in w_4, and hence their preparatory transitions are t_{22}, and $t_{22}[o_{12}]$ and $t_{22}[o_{22}]$ is hold. Then the complete transition sequence for individual I_1 is $\alpha(I_1) = (t_{11}, t_{22}, t_{23}, t_{11}, t_{31}, t_{22}, t_{23}, t_{11}, t_{12}, t_{32}, t_{31}, t_{24}, t_{32}, t_{33}, t_{24}, t_{13}, t_{33}, t_{14}, t_{14}, t_{14}, t_{34}, t_{34})$.

2) AMENDING

According to the above encoding and decoding method, individual I corresponds to the unique transition sequence $\alpha(I)$. Although $\alpha(I)$ includes the number of transitions required from M_0 to M_f, but $\alpha(I)$ itself may not be feasible. It may not be fired in order, and/or cause deadlock. Thus, the modification of such I or $\alpha(I)$ is necessary. For example, consider individual I_1 and its corresponding transition sequence $\alpha(I_1)$ in Example 2. Let $\alpha(I_1) = \sigma_1\sigma_2$ where $\sigma_1 = t_{11}t_{22}t_{23}t_{11}t_{31}t_{22}t_{23}t_{11}t_{12}$, and $M_0[\sigma_1 > M_1$. Then $M_1 \neq M_f$, and under M_1, all transitions are dead. Thus $\alpha(I_1)$ is not feasible. Thus, the feasibility of each individual should be checked and the infeasible individuals are translated into feasible ones. In this article, the detection and amending algorithm (Algorithm DA) proposed in [25] is embedded in a decoding process to obtain the feasible sequence of firing transitions from M_0 to M_f. The reader can refer to [25] for more details.

B. FITNESS FUNCTION

The fitness function could be used to guide the EDA. In this article, it is the makespan, i.e., the completion time of the last job.

For an individual and its complete and feasible sequence of transitions $\alpha(I) = t_0t_1t_2\ldots t_{L-1}$, let $M_k[t_k > M_{k+1}$, $k = 0, 1, \ldots, L-1$, and $f(t_k[o_{ij}])$ denote the firing time of t_k, i.e., when job J_i enters workstation $w(o_{ij})$. Since there is only one machine that can process jobs in $w(o_{ij})$, job J_i cannot be processed immediately after transition t_k fires when the machine is busy.

Let $s(o_{ij})$ denote the start time of operation o_{ij}, the j-th operation of J_i, that is, for example, the firing time of transitions t_{111} or t_{331} in Figure 1. Then, $f(t_k[o_{ij}]) \leq s(oij)$, and $s(o_{ij}) = f(t_k[o_{ij}])$ only if the machine is idle when job J_i enters workstation $w(o_{ij})$.

For J_i, $t_k[o_{ij}]$ can be fired only after operation $o_{i(j-1)}$ is finished. Hence, $f(t_k[o_{ij}]) \geq s(o_{i(j-1)}) + d(o_{i(j-1)})$. On the other hand, the transitions in $\alpha(I)$ are sequentially fired, and the firing time of $t_k[o_{ij}]$ should be after the firing of t_{k-1}, i.e., $f(t_k[o_{ij}]) \geq f(t_{k-1})$. Then, we have

$$f(t_k[o_{ij}]) = max\{s(o_{i(j-1)}) + d(o_{i(j-1)}), f(t_{k-1})\} \tag{1}$$

If there are no other jobs in workstation $w(o_{ij})$ when job J_i enters it for its operation o_{ij}, job J_i can start its operation o_{ij}, i.e., the start time of o_{ij} is

$$s(o_{ij}) = f(t_k[o_{ij}]) \tag{2}$$

If there exist other jobs in $w(o_{ij})$ that arrive earlier than job J_i, then the start time of o_{ij} is not earlier than the completion time of any of their operations that have already been started at $f(t_k[o_{ij}])$ or before. That is, the start time of o_{ij} is

$$\begin{aligned} s(o_{ij}) &= max\{s(o_{mn}) + d(o_{mn}), f(t_k[o_{ij}])|w(o_{mn}) \\ &= w(o_{ij}) \; and \; f(t_l[o_{mn}]) \leq f(t_k[o_{ij}])\} \end{aligned} \tag{3}$$

By (1)-(3), the firing time of every transition and the start time of every operation can be computed recursively. The fitness function of individual I is obtained

$$f(I) = f(t_{L-1}).$$

C. INITIALIZATION

The initial population can be randomly generated. For generating an individual $I = (S_o; S_r)$, first randomly create a permutation with repetition of jobs in which each type-q job appears $\chi(q)$ times, and then, randomly select a route for each job from its route set. Through Algorithm DA, the infeasible individuals are amended into feasible ones. Let Ω denote the current population of feasible individuals.

Decoding each individual in Ω and calculating their fitness values. Sort all individuals in Ω according to the ascending order of their fitness values, and select the best $\rho \times |\Omega|(\rho \in (0, 1])$ individuals as the elite set, denoted as Ω_e. In the simulation of this example, the ρ value is obtained by using an experimental optimization approach, and $\rho = 0.3$.

D. PROBABILISTIC MODEL AND GENERATING NEW INDIVIDUALS

The probabilistic model represents a main issue for EDA and its performance is closely related to it [32]. The best choice of a model is crucial. On the other hand, the efficiencies of model constructing and information sampling are closely related to the performance of the algorithm. Hence, the choice of the probabilistic model plays an important role in EDA's success.

In this article, the probability model is designed as a dominance matrix. It is based on the global statistic information from the elite set. Since an individual contains two parts: operation and route, two kinds of probability sub-models are constructed, i.e., the operation probability model Π, and the route probability model Ξ. They are used together to construct a new individual. Such two probability models are first constructed as follows.

Let $\Pi_1 = (a_{ij})$ be an $n \times L$ matrix where n is the number of jobs to be processed and L is the length of S_o. Let γ_{ij} denote the number of individuals in Ω_e, in which job J_i is at position j of their operation parts, and $a_{ij} = \gamma_{ij}/|\Omega_e|$).

In $\Pi_1 = (a_{ij})$ defined above, each individual in Ω_e is treated equally. In order to distinguish their importance, in this article, we set a weight for each individual according to their fitness values.

Let I_w and I_b be the worst and best individuals in current elite set Ω_e. The weight of individual $I \in \Omega_e$ is defined as $g(I) = (f(I_w) - f(I) + 1)/f(I_b)$. It is obvious that individuals with smaller fitness values or makespans are given larger weights. With this weight function $g(I)$ and $\Pi_1 = (a_{ij})$ defined above, we can design our operation probability matrix Π and route probability matrix Ξ. Π has the same dimension as Π_1 and can be obtained by modifying Π_1, and Ξ is an $n \times K$ matrix, where K is the total number of all different processing routes for all jobs.

The detailed algorithm for constructing probability Models Π and Ξ is given in algorithm CPM.

In the above **Algorithm CPM**, the operation probability matrix Π and route probability matrix Ξ are determined by the voting method of elite individuals in Ω_e. Although each elite individual in Ω_e participates in voting, the better the individual is, the greater the contribution to Π and Ξ.

With probability models Π and Ξ, we can construct new individuals. Two methods for constructing new individuals are proposed in this article. In them, element a_{ij} in Π is considered as the probability of selection of job J_i in the j-th position. In the first one, we first extend $n \times L$ matrix Π to $L \times L$ matrix Π_{ext} so that a row of Π corresponding to job J_i is $\chi(J_i)$ rows of Π_{ext} in succession. Then, according to the descending order of elements in Π_{ext} and for maximum element a_{ij}, set job i to position j in the sequence of a new individual, and then set all elements on the row and column in which a_{ij} is into -1. Repeat until all the elements of Π_{ext} are -1.

In the second method, selecting job for each position and a route for a job in the new individual is done by roulette based on Π and Ξ. The details of algorithms called CNI1 and CNI2 where CN represents Constructing New-individuals are as follows.

Algorithm CPM //Constructing Probabilistic Models

Input: Ω_e;
Output: Π and Ξ; Begin

```
1: At the beginning, set $\Pi \equiv (a_{ij})_{n\times L} = \mathbf{0}$; $B \equiv (b_{ij})_{n\times L} = \mathbf{0}$; $\Xi \equiv (c_{ik})_{n\times K} = \mathbf{0}$; $D \equiv (d_{ik})_{n\times K} = \mathbf{0}$;
2: for ($i = 1; i \leq n; i{+}{+}$){ // $J = (J_1, J_2, \ldots, J_n)$ is given.
3:   for (each $I \in \Omega_e$){ // $I = (S_o, S_r)$ where $S_o = (p_0, \ldots, p_{L-1})$ and $S_r = (\sigma_1, \sigma_2, \ldots, \sigma_n)$.
4:   $g(I) = [f(I_w) - f(I) + 1]/f(I_b)$;
5:     for ($l = 1; l \leq L; l++$){;
6:       if ($p_l = J_i$){$a_{il} := a_{il} + 1$; $b_{il} := b_{il} + g(I)$; };
7:     }end for ($l$);
8:     for ($k = 1; k \leq K; k++$){ // for $\pi_j$ and $\pi = (\pi_1, \pi_2, \ldots, \pi_K)$ is given;
9:       if ($\sigma_i = \pi_k$){$c_{ik} := c_{ik} + 1$; $d_{ik} := d_{ik} + g(I)$; };
10:    }end for ($k$)
11:  }end for (each $I$)
12:}end for($i$)
13:for ($j = 1; j \leq L; j++$){
14:  $\Pi_j := \Pi_j/|\Omega_e|$; // $\Pi$ normalization; $\Pi_j$ is the $j$-th column of $\Pi$.
15:  Let $b_j = \sum_i b_{ij}$; $B_j := B_j/b_j$; // $B_j$ is the $j$-th column of $B$.
16:  $\Pi := (\Pi + B)/2$; // $\Pi$ operation probability matrix.
17: } end for ($j$)
18: for ($i = 1; i \leq n; i++$){
19:  Let $c_i = \sum_j c_{ij}$; $\Xi_i := \Xi_i/c_i$; // $\Xi$ normalization; $\Xi_i$ is the $i$-th row of $\Xi$.
20:  Let $d_i = \sum_j d_{ij}$; $D_i := D_i/d_i$; // $D$ normalization; $D_i$ is the $i$-th row of $D$.
21:  $\Xi := (\Xi + D)/2$; // $\Xi$ route probability matrix.
22:}end for ($i$)
23:Output $\Pi$ and $\Xi$;
End
```

E. LOCAL SEARCH

To balance the global and local searches and to further improve the performance, a simplified simulated annealing (SSA) algorithm is employed as the local search. Note that simulated annealing has been successfully combined with other intelligent optimization methods. SSA is acted on new

Algorithm CNI1 // Constructing a New Individual $I = (S_o; S_r)$ Through Π and Ξ

```
Input: $\Pi$ and $\Xi$;
Output: $I = (S_o; S_r)$;  Begin
    1: Let $S_o$ and $S_r$ be $1 \times L$ and $1 \times n$ empty arrays;
    2: Let $\Pi_{ext}$ be the extended matrix of $\Pi$; Let $\Xi_1 = \Xi$;
    3: for ($count = 0$; $count < L$; $count++$){ // Constructing $S_o$ through $\Pi_{ext}$.
    4:   1.1) Find an element $e_{uj}$ with the maximum value in $\Pi_{ext}$; // (select the first one if there is more than one such element);
    5:   1.2) If the row, in which $e_{uj}$ is located, corresponds to job $J_i$, add job $J_i$ at position $j$ in $S_o$;
    6:   1.3) Set all elements in row $u$ and column $j$ of $\Pi_{ext}$ as -1;
    7: } end for // $S_o$ is constructed;
    8: for ($count = 0$; $count < n$; $count++$){ // Constructing route $S_r$ for $S_o$ through $\Xi$
    9:   2.1) find an element $c_{ij}$ with the maximum value in $\Xi$;
    10:  2.2) add route $\pi_j$ at position $i$ in $S_r$, that is, set $\pi_j$ as the route of job $J_i$ in $S_r$;
    11:  2.3) Set all elements in row $i$ of $\Xi$ as -1;
    12:}end for // $S_r$ is constructed;
    Output: $I = DA(S_o; S_r)$; // A new feasible individual $I$ is constructed.
End
```

Algorithm CNI2 //Constructing a New Individuals Through Π and Ξ

```
// constructing $S_o$ through $\Pi \equiv (a_{ij})$;
1: Let $S_o$ and $S_r$ be $1 \times L$ and $1 \times n$ empty arrays;
2: Let $\Pi_1 = \Pi$; Let $\Xi_1 = \Xi$;
3: Let A[$n$] = $(0, 0, \ldots, 0)$; // A[$j$] is the number of $J_j$ appearing in $S_o$;
4: for ($i = 0$; $i < L$; $i++$){
5:   $\mu$ =rand(0, 1);
6:   for ($j = 1$; $j \le n$; $j++$){
7:     if($\mu \le \sum_{k=1,\ldots,j} a_{ik}$ && A[$j$] $< \chi(J_j)$){ // Select a job for position $i$ by Roulette;
8:       add job $J_j$ at position $i$ in $S_o$; A[$j$] $\leftarrow$A[$j$] + 1; break;
9:     }
10:  }end for($j$)
11: } end for($i$)
// Constructing route $S_r$ for $S_o$ through $\Xi$.
12:for ($i = 0$; $i < n$; $i++$){
13: $\mu$ =rand(0, 1); // generating a random number.
14: for ($j = 0$; $j < K$; $j++$){
15:   if ($\mu \le \sum_{s=1,\ldots,j} c_{is}$){ // Select a route for job $i$ by Roulette.
16:     add route $\pi_j$ at position $i$ in $S_r$, i.e., set $\pi_j$ as the route of job $J_i$ in $S_r$; break;}
17: }end for($j$)
18:} end for($i$) // $S_r$ is constructed;
19:Output $I = DA(S_o; S_r)$; //A new feasible individual $I$ is constructed.
End
```

individual with a probability $p_\alpha (p_\alpha \in (0, 1])$. Its parameter temperature, *Temp*, is supposed to be constant. If individual I is selected to execute local search, $Temp = f(I)/(U \times n)$ where U is a constant. Its termination condition is that the maximum number of iterations, i_{max}, is reached or I has been improved for x times. In the simulation, we set $i_{max} = 30$ and $x = 3$. A new individual is produced by the so-called Neighbor Search (NS), in which three operations, job-insert, job-swap, and route-change, are executed sequentially. These three operations are defined as follows.

Job-insert: Randomly choose two different jobs from the operation part S_o of I, and then insert the back one before the front one.

Job-swap: Randomly select two different jobs J_u and J_v from the operation part S_o of I, and then swap them.

Route-change: Randomly select a job with multiple routes, and then change its current route in route part S_r of I to any one of its other routes.

Algorithm SSA is shown as follows.

F. PROPOSED HYBRID ESTIMATION OF DISTRIBUTION ALGORITHM

With the above design, we can propose the HEDA procedure for solving flexible job shop manufacturing environment with limited capacity buffers as follows. It contains two main parts in every generation. At global exploration, a probability model is built with the elite individuals of an entire population to generate new individuals. At local search, the newly generated individuals adopt in probability p_α multiple local search operators based on the problem characteristics for further exploitation. The algorithm stops when the maximum number of generations G_{max} is reached.

If p_α is set to 0, the local search SSA is not called in every generation, and the algorithm is simplified into so-called basic EDA.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

FJS with limited buffers is a new scheduling problem that is first attempted to be studied in this article. In current literature, there are no any algorithms and benchmarks to be developed for it. As a result, except for the algorithms proposed in this article, it is unrealistic to carry out a wider comparison. In this section, the proposed HEDA is tested by a simple FJS with limited buffers, which consists of nine workstations $w_1 - w_9$. Each of w_2, w_3, w_4, w_6, and w_7 has a buffer with capacity 2 while w_1, w_5, w_8 and w_9 each have a buffer with capacity 1. The considered FJS can process three types of jobs, types-q_1, q_2, and q_3. Its PN model is shown in Figure 2. There are $\varphi(q_i) = n_i$, $i \in Z_3$, type-q_i jobs to be processed, and $n = n_1 + n_2 + n_3$. O$(\pi_1) = p_{1s}$ $t_{11}p_{111}t_{111}p_{112}t_{112}p_{113}t_{12}p_{121}t_{121}p_{122}t_{122}p_{123}t_{13}p_{131}t_{131}p_{132}$ $t_{132}p_{133}t_{14}p_{141}t_{141}p_{142}t_{142}p_{143}t_{15}p_{151}t_{151}p_{152}t_{152}p_{153}t_{16}p_{1e}$,

Algorithm SSA(I)

Input: I, I_0 // I and I_0 are individuals and I_0 is the best one in the current population Ω;

Output: Ind_0; Begin

```
1:$Ind_0 \leftarrow I$; $Ind_1 \leftarrow I$;
2:$Temp = f(I)/(U \times n)$; $u = v = 0$;
3: While (the termination condition is not satisfied){
4:  $Ind_1 \leftarrow NS(Ind_1)$, $Ind_1 \leftarrow DA(Ind_1)$;
5:  If($f(Ind_1) < f(Ind_0)$){
6:    $Ind_0 \leftarrow Ind_1$; $v \leftarrow v+1$;
7: } Else {
8:  If ((rand(0, 1) $\leq$ exp($-(f(Ind_1) - (f(Ind_0))/Temp$){
9:    $Ind_0 \leftarrow Ind_1$; //accept worse individual;
10: }Else{
11:   $Ind_2 \leftarrow NS(I_0)$, $Ind_2 \leftarrow DA(Ind_2)$;
12:   If ($f(Ind_2) < f(Ind_0)$){ $Ind_0 \leftarrow Ind_2$}}
13: } //end Else
14:  $Ind_1 \leftarrow Ind_0$;
15:  $u \leftarrow u+1$; //end Else
16:} end while
17:Return $DA(Ind_0)$; // Return an feasible individual.
```
End

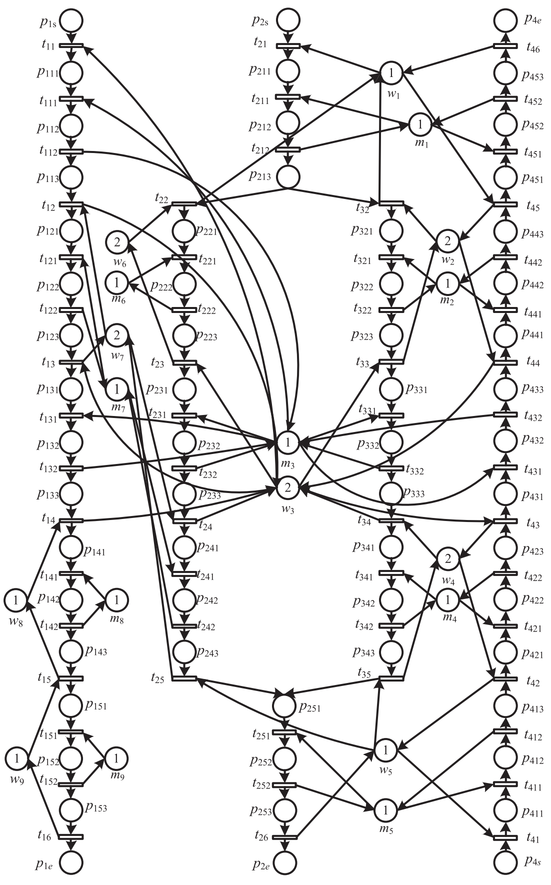


FIGURE 2. PNS of the FJS.

and $O(\pi_4) = p_{4s}t_{41}p_{411}t_{411}p_{412}t_{412}p_{413}t_{42}p_{421}t_{421}p_{422}t_{422}p_{423}t_{43}p_{431}t_{431}p_{432}t_{432}p_{433}t_{44}p_{441}t_{441}p_{442}t_{442}p_{443}t_{45}p_{451}t_{451}p_{452}t_{452}p_{453}t_{46}p_{4e}$, respectively. Type-$q_2$ jobs have two

Algorithm HEDA

```
1:Initial parameters $G_{max}$, $N_{pop}$, $N_{esc}$, $N_{New}$, and $p_\alpha$;
2:Randomly generate initial population $\Omega$;
3: Sorting $\Omega$ // according to in ascending order of individual fitness values, and denote $\Omega$ as $\Omega = \{I_0, I_1, \ldots, I_{Npop}\}$;
4: select $\Omega_e = \{I_0, I_1, \ldots, I_{Nesc-1}\}$; // $I_0$ is the best individual in $\Omega$;
5: For ($g = 0$; $g < G_{max}$; $g++$){
6:  Let($\Pi$, $\Xi$) = CPM($\Omega_e$); // establish $\Pi$ and $\Xi$ from $\Omega_e$.
7:  $Ind_0$ = CNI1($\Pi$, $\Xi$);
8:  For($i = 1$; $i < N_{new}$; $i++$){
9:    $Ind_i$ = CNI2($\Pi$, $\Xi$)
10:    If ($rand(0, 1) < p_\alpha$){
11:      $Ind_i$ = SSA($Ind_i$); }
12:      Add $Ind_i$ to $\Omega_1$;
13:   } // end For($i$)
14:   Sorting $\Omega \cup \Omega_1$; select the first $N_{esc}$ best individuals as $\Omega_e$, $\Omega_e = \{I_0, I_1, \ldots, I_{Nesc-1}\}$; // the best individual is denoted as $I_0$.
15: }end For($g$)
16:output $I_0$;
```
End

TABLE 1. Job numbers in different instances.

Instances	n_1	n_2	n_3
In01	2	3	2
In02	3	5	4
In03	5	7	6
In04	7	7	8
In05	8	10	10
In06	10	10	10
In07	12	10	10
In08	12	12	10
In09	12	12	12
In10	14	12	14
In11	16	14	14
In12	16	16	16
In13	16	18	16
In14	18	18	18
In15	18	18	20
In16	20	20	20

processing routes, $O(\pi_2) = p_{2s}t_{21}p_{211}t_{211}p_{212}t_{212}p_{213}t_{22}p_{221}t_{221}p_{222}t_{222}p_{223}t_{23}p_{231}t_{231}p_{232}t_{232}p_{233}t_{24}p_{241}t_{241}p_{242}t_{242}p_{243}t_{25}p_{251}t_{251}p_{252}t_{252}p_{253}t_{26}p_{2e}$ or $O(\pi_3) = p_{2s}t_{21}p_{211}t_{211}p_{212}t_{212}p_{213}t_{32}p_{321}t_{321}p_{322}t_{322}p_{323}t_{33}p_{331}t_{331}p_{332}t_{332}p_{333}t_{34}p_{341}t_{341}p_{342}t_{342}p_{343}t_{35}p_{251}t_{251}p_{252}t_{252}p_{253}t_{26}p_{2e}$. Sixteen instances with different numbers of jobs, (n_1, n_2, n_3), are designed to evaluate the performance of the proposed HEDA, and they are shown in Table 1. In these instances, we suppose that job processing time is randomly distributed in the range [4, 40], and the detailed values are shown in Table 2.

TABLE 2. Processing time for operations.

Type-q_1	Type-q_2			Type-q_3
π_1	π_2		π_3	π_4
o_{11}:8		o_{21}:4		o_{41}:5
o_{12}:24	o_{22}:23		o_{32}:32	o_{42}:22
o_{13}:5	o_{23}:8		o_{33}:6	o_{43}:4
o_{14}:21	o_{24}:38		o_{34}:20	o_{44}:17
o_{15}:4		o_{25}:5		o_{45}:6

TABLE 3. Parameter values of different factor level.

Parameters	Factor level		
	1	2	3
ρ	0.1	0.3	0.5
N_{new}	10	20	30
p_α	0.02	0.04	0.08
U	14	16	18

A. PARAMETER DETERMINATION

In the simulation calculation, the first two parameters of Algorithm HEDA, population size $|\Omega|$ and maximum number of generations G_{max}, are set as $|\Omega| = 100$ and $G_{max} = 50 \times n$, respectively. The other four parameters, the elite individual percentage ρ, the number of new individuals N_{new}, the probability of acting SSA on a new individual p_α, and temperature constant U, will be determined by using the Taguchi method [36].

Parameter ρ determines directly the size of the elite set that provides information for the probability model. If it is too large, some poor individuals are included and have bad effect on the probability model; while if it is too small, the population may mature too early. Parameter N_{new} determines the size of the seed set. A small N_{new} leads to obtain only a few good genes and has a too fast convergence speed while a large value implies that poor individuals could be involved in. Parameter p_α is the probability of local search conducted on new individuals, which balances the global search and the local search. Temperature *Temp* in SSA has influence on the probability accepting a worse solution. It is proportional to a constant U, i.e., $Temp = f(I)/(U \times n)$. The value domains these four parameters are set as $\rho \in \{0.1, 0.3, 0.5\}$, $N_{new} \in \{10, 20, 30\}$, $p_\alpha \in \{0.02, 0.04, 0.08\}$, and $U \in \{14, 16, 18\}$, respectively. For each parameter, three factor levels are used and listed in Table 3, and hence, the orthogonal array $L_9(3^4)$ is chosen. For each parameter combination of $L_9(3^4)$, HEDA is run 10 times independently regarding instance In03. The makespan average in 10 run times is used as the response variables (RV). The orthogonal array and the values of response variables are listed in Table 4, and then the statistic results are summarized in Table 5.

TABLE 4. Orthogonal array $L_9(3^4)$ and response variable values.

Trail	Factor level				
	ρ	N_{new}	p_α	U	RV
1	1	1	1	1	277.1
2	1	2	2	2	264.5
3	1	3	3	3	260.8
4	2	1	2	3	266.8
5	2	2	3	1	261.5
6	2	3	1	2	271.7
7	3	1	3	2	270.8
8	3	2	1	3	275.8
9	3	3	2	1	260.1

TABLE 5. Statistic results.

Factor level	RV			
	ρ	N_{new}	p_α	U
1	267.47	271.57	274.87	266.23
2	266.67	267.27	263.8	269
3	268.90	264.20	264.37	267.8
Delta	2.23	7.37	11.07	2.77
Rank	4	2	1	3

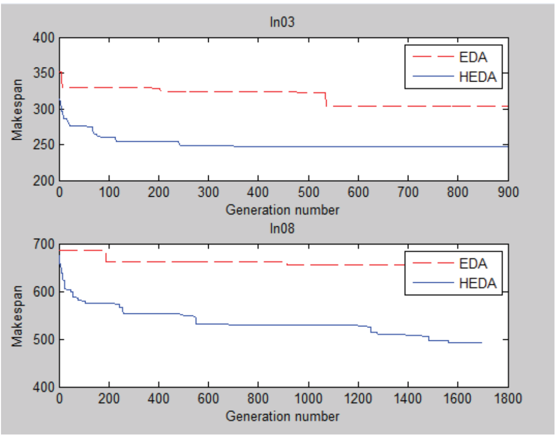


FIGURE 3. The convergence tendency of EDA and HEDA with makespan on In03 and In08.

From Table 5, it can be seen that p_α has the most significant impact on HEDA performance. N_{new} ranks the second, U the third, and ρ the last. According to their corresponding factor levels (the bold face), the suggested parameter set-up for HEDA are determined as $\rho = 0.3$, $N_{new} = 30$, $p_\alpha = 0.04$, and $U = 14$.

B. EXPERIMENTAL RESULTS

In order to test the effectiveness of our proposed algorithms and explain the improvement effect of SSA on EDA, Algorithms EDA and HEDA are run 10 times independently on

TABLE 6. Comparison of simulation results and runtime of EDA and HEDA.

Instances	*EDA*				*HEDA*			
	BST	TIME/B	AVG	TIME/A	BST	TIME/B	AVG	TIME/A
In01	118	2	124.5	2.0	**108**	4	**109.2**	4.9
In02	196	12	208.9	12.0	**174**	26	**179.5**	26.1
In03	303	44	312.0	43.1	**247**	93	**264.2**	100.5
In04	378	76	389.9	79.3	**295**	183	**327.3**	177.4
In05	490	162	505.6	169.2	**402**	374	**422.3**	376.8
In06	540	184	558.1	198.0	**406**	445	**452.7**	444.3
In07	597	245	608.2	245.6	**453**	542	**500.3**	503.0
In08	655	270	666.5	282.8	**491**	656	**528.0**	655.2
In09	667	336	695.3	347.6	**495**	748	**542.3**	755.3
In10	747	467	777.6	478.5	**578**	997	**620.1**	1025.7
In11	845	581	867.8	640.8	**668**	1308	**714.6**	1259.9
In12	922	887	949.1	833.0	**714**	1696	**754.6**	1643.2
In13	980	883	993.8	862.5	**759**	1778	**806.3**	1794.4
In14	1049	978	1079.5	1126.5	**805**	2326	**861.1**	2313.9
In15	1082	1470	1107.2	1323.7	**800**	2622	**886.5**	2565.5
In16	1173	1383	1210.9	1490.1	**893**	3273	**964.6**	2993.3

all 16 instances and their experimental results are compared. Their average and best (or minimum) makespan values are listed in Table 6, where BST denotes the best makespan among 10 trials, TIME/B denotes the runtime of a trial through which the best makespan is obtained, and AVG and TIME/A denote the average makespan and average runtime of 10 trials, respectively.

From Table 6,we know that two algorithms can give feasible solutions for each instance in a relatively short time. The shortest time is only a few seconds, and the longest average time is 2993.3 seconds of HEDA for In16. For each instance, the makespan results obtained by HEDA are superior to those obtained by the basic EDA. This shows that the embedding SSA has greatly improved the performance of EDA. The larger the scale of the problem, the greater the improvement. For example, the performance improvement for average makespans of In01, In08, In09, and In16 are (124.5-109.2)/124.5 = 12.3%, 20.8%, 22%, and 20.3%, respectively. Of course, these improvements are time-consuming. Thus, it is better to run algorithm HEDA in order to obtain at the better schedules at the expense of more time.

The convergence tendency of EDA and HEDA of In03 and In08 with makespan is shown in Figure 3. It can be seen that EDA is easy to mature early, and converges faster than HEDA. It also shows that SSA plays a certain role in overcoming the early maturing phenomenon and HEDA owns better global searching ability. HEDA's makespan is improving globally and constantly, especially for In08 and hence, it is reasonable and necessary to increase the maximum iteration times as the scale of the problem increases, i.e., $G_{max} = 50 \times n$. For In08, an optimal or suboptimal schedule with makespan 491 is obtained by HEDA.

V. CONCLUSION

In this article, the production scheduling problem of FJS with limited buffers is studied. Based on the place-timed Petri net model of the considered system, a novel hybrid estimation of distribution algorithm (HEDA) is proposed to minimize the makespan. A candidate solution is coded as an individual that consists of two sections. In the first section, a processing route is set for each job; and the second part is a possible sequence of operations. Under the monitoring of a deadlock controller, such an individual can be decoded as a feasible schedule. Corresponding to different sections of an individual, two probability models, route and operation probability models, are established via a voting procedure in which individual weighted differences are considered. Based on these models, an effective procedure for constructing offspring individuals is proposed. For each new individual, a simple local search, SSA, is performed with a certain probability in order to improve the performance of EDA. Simulation results show the effectiveness of the proposed method over EDA.

This work is of important practice significance for job-shop scheduling with limited buffers. Since no existing comparable results are available, more studies for establishing more effective methods are needed in the future. On the other hand, there are many evolutionary algorithms that may be used to solve the problem considered here. But for each different algorithm, as done in this article, we need to establish the corresponding encoding and decoding scheme, the relevant operators, and the parameter setting scheme, and how to improve the performance of the algorithm by embedding appropriate local searches. All these are worthy of further study.

REFERENCES

[1] I. A. Chaudhry and A. A. Khan, "A research survey: Review of flexible job shop scheduling techniques," *Int. Trans. Oper. Res.*, vol. 23, no. 3, pp. 551–591, May 2016.

[2] M. Pinedo, *Scheduling: Theory, Algorithms and Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2008.

[3] O. Holthaus and C. Rajendran, "A study on the performance of scheduling rules in buffer-constrained dynamic flowshops," *Int. J. Prod. Res.*, vol. 40, no. 13, pp. 3041–3052, Jan. 2002.

[4] B. Liu, L. Wang, and Y.-H. Jin, "An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers," *Comput. Oper. Res.*, vol. 35, no. 9, pp. 2791–2806, Sep. 2008.

[5] Q.-K. Pan, L. Wang, L. Gao, and W. D. Li, "An effective hybrid discrete differential evolution algorithm for the flow shop scheduling with intermediate buffers," *Inf. Sci.*, vol. 181, no. 3, pp. 668–685, Feb. 2011.

[6] C. H. Papadimitriou and P. C. Kanellakis, "Flowshop scheduling with limited temporary storage," *J. ACM*, vol. 27, no. 3, pp. 533–549, Jul. 1980.

[7] B. Qian, L. Wang, D.-X. Huang, W.-L. Wang, and X. Wang, "An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers," *Comput. Oper. Res.*, vol. 36, no. 1, pp. 209–233, Jan. 2009.

[8] C. Zhang, Z. Shi, Z. Huang, Y. Wu, and L. Shi, "Flow shop scheduling with a batch processor and limited buffer," *Int. J. Prod. Res.*, vol. 55, no. 11, pp. 3217–3233, Jun. 2017.

[9] P. Brucker, S. Heitmann, J. Hurink, and T. Nieberg, "Job-shop scheduling with limited capacity buffers," *OR Spectr.*, vol. 28, no. 2, pp. 151–176, Apr. 2006.

[10] A. D'Ariano, D. Pacciarelli, and M. Pranzo, "A branch and bound algorithm for scheduling trains in a railway network," *Eur. J. Oper. Res.*, vol. 183, no. 2, pp. 643–657, Dec. 2007.

[11] S. A. Fahmy, T. Y. ElMekkawy, and S. Balakrishnan, "Deadlock-free scheduling of flexible job shops with limited capacity buffers," *Eur. J. Ind. Eng.*, vol. 2, no. 3, pp. 231–252, Feb. 2008.

[12] S. A. Fahmy, T. Y. ElMekkawy, and S. Balakrishnan, "Mathematical formulations for scheduling in manufacturing cells with limited capacity buffers," *Int. J. Oper. Res.*, vol. 7, no. 4, pp. 463–486, Apr. 2010.

[13] M. C. Gomes, A. P. Barbosa-Póvoa, and A. Q. Novais, "Optimal scheduling for flexible job shop operation," *Int. J. Prod. Res.*, vol. 43, no. 11, pp. 2323–2353, Jun. 2005.

[14] S. Heitmann, "Job-shop scheduling with limited buffer capacities," Ph.D. dissertation, Dept. Math./Inform., Univ. Osnabrück, Osnabrück, Germany, 2007.

[15] S. Q. Liu, E. Kozan, M. Masoud, Y. Zhang, and F. T. S. Chan, "Job shop scheduling with a combination of four buffering constraints," *Int. J. Prod. Res.*, vol. 56, no. 9, pp. 3274–3293, May 2018.

[16] M. Pranzo and D. Pacciarelli, "An iterated greedy metaheuristic for the blocking job shop scheduling problem," *J. Heuristics*, vol. 22, no. 4, pp. 587–611, Aug. 2016.

[17] M. Sama, A. D'Ariano, P. D'Ariano, and D. Pacciarelli, "Scheduling models for optimal aircraft traffic control at busy airports: Tardiness, priorities, equity and violations considerations," *Omega*, vol. 67, pp. 81–98, Mar. 2017.

[18] H. Toba, "A tight flow control for job-shop fabrication lines with finite buffers," *IEEE Trans. Autom. Sci. Eng.*, vol. 2, no. 1, pp. 78–83, Jan. 2005.

[19] A. Witt and S. Vo beta, "Job shop scheduling with buffer constraints and jobs consuming variable buffer space," in *Proc. Int. Heinz Nixdorf Symp.* Berlin, Germany: Springer, 2010.

[20] M. P. Fanti and M. Zhou, "Deadlock control methods in automated manufacturing systems," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 34, no. 1, pp. 5–22, Jan. 2004.

[21] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on Petri nets—A literature review," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 4, pp. 437–462, Jul. 2012.

[22] Y. Feng, K. Xing, H. Liu, and Y. Wu, "Two-stage design method of robust deadlock control for automated manufacturing systems with a type of unreliable resources," *Inf. Sci.*, vol. 484, pp. 286–301, May 2019.

[23] N. Wu and M. Zhou, "Deadlock resolution in automated manufacturing systems with robots," *IEEE Trans. Autom. Sci. Eng.*, vol. 4, no. 3, pp. 474–480, Jul. 2007.

[24] K. Xing, M. Chu Zhou, H. Liu, and F. Tian, "Optimal Petri-net-based polynomial-complexity deadlock-avoidance policies for automated manufacturing systems," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 39, no. 1, pp. 188–199, Jan. 2009.

[25] K. Xing, L. Han, M. Zhou, and F. Wang, "Deadlock-free genetic scheduling algorithm for automated manufacturing systems based on deadlock control policy," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 42, no. 3, pp. 603–615, Jun. 2012.

[26] R. Leisten, "Flowshop sequencing problems with limited buffer storage," *Int. J. Prod. Res.*, vol. 28, no. 11, pp. 2085–2100, Nov. 1990.

[27] J.-Q. Li and Q.-K. Pan, "Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm," *Inf. Sci.*, vol. 316, pp. 487–502, Sep. 2015.

[28] R. S. Hansmann, T. Rieger, and U. T. Zimmermann, "Flexible job shop scheduling with blockages," *Math. Methods Oper. Res.*, vol. 79, no. 2, pp. 135–161, Apr. 2014.

[29] J. Lange and F. Werner, "Approaches to modeling train scheduling problems as job-shop problems with blocking constraints," *J. Scheduling*, vol. 21, no. 2, pp. 191–207, Apr. 2018.

[30] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *Eur. J. Oper. Res.*, vol. 143, no. 3, pp. 498–517, Dec. 2002.

[31] Y. Mati, C. Lahlou, and S. Dauzère-Pérès, "Modelling and solving a practical flexible job-shop scheduling problem with blocking constraints," *Int. J. Prod. Res.*, vol. 49, no. 8, pp. 2169–2182, Apr. 2011.

[32] P. Larra naga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.*, Boston, MA, USA: Kluwer, 2001.

[33] B. Jarboui, M. Eddaly, and P. Siarry, "An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems," *Comput. Oper. Res.*, vol. 36, no. 9, pp. 2638–2646, Sep. 2009.

[34] M. C. Zhou and K. Venkatesh, *Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach*. Singapore: World Scientific, 1998.

[35] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 4, pp. 904–916, Jul. 2019.

[36] D. C. Montgomery, *Design and Analysis of Experiments*. Hoboken, NJ, USA, Wiley, 2005.

[37] N. Qi Wu and M. Zhou, "Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 2, pp. 446–454, Apr. 2012.

[38] L. Bai, N. Wu, Z. Li, and M. Zhou, "Optimal one-wafer cyclic scheduling and buffer space configuration for single-arm multicluster tools with linear topology," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 46, no. 10, pp. 1456–1467, Oct. 2016.

[39] Q. Zhu, Y. Qiao, and N. Wu, "Optimal integrated schedule of entire process of dual-blade multi-cluster tools from start-up to close-down," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 2, pp. 553–565, Mar. 2019.

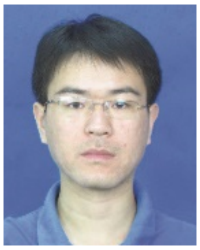
[40] Q. Zhu, Y. Qiao, N. Wu, and Y. Hou, "Post-processing time-aware optimal scheduling of single robotic cluster tools," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 2, pp. 597–605, Mar. 2020.

[41] Y. Hou, N. Wu, Z. Li, Y. Zhang, T. Qu, and Q. Zhu, "Many-objective optimization for scheduling of crude oil operations based on NSGA-III with consideration of energy efficiency," *Swarm Evol. Comput.*, vol. 57, Sep. 2020, Art. no. 100714.

[42] J. Luo, Z. Liu, S. Wang, and K. Xing, "Robust deadlock avoidance policy for automated manufacturing system with multiple unreliable resources," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 3, pp. 812–821, May 2020.

[43] S. Wang, W. Duo, X. Guo, X. Jiang, D. You, K. Barkaoui, and M. Zhou, "Computation of an emptiable minimal siphon in a subclass of Petri nets using mixed-integer programming," *IEEE/CAA J. Autom. Sinica*, early access, Jun. 2, 2020, doi: 10.1109/JAS.2020.1003210.

[44] B. Huang, M. Zhou, C. Wang, A. Abusorrah, and Y. Al-Turki, "Deadlock-free supervisor design for robotic manufacturing cells with uncontrollable and unobservable events," *IEEE/CAA J. Autom. Sinica*, early access, Jun. 2, 2020, doi: 10.1109/JAS.2020.1003207.



ZHENXIN GAO received the B.S. degree in automation science and technology from the Xi'an University of Posts and Telecommunications, Xi'an, China, in 2010. He is currently pursuing the Ph.D. degree with the Systems Engineering Institute, Xi'an Jiaotong University, Xi'an.

His research interest includes control and scheduling of automated manufacturing systems.

YANXIANG FENG received the B.S. degree in automation science and technology and the Ph.D. degree in systems engineering from Xi'an Jiaotong University, Xi'an, China, in 2010 and 2017, respectively.

He is currently with the State Key Laboratory for Manufacturing Systems Engineering, Xi'an Jiaotong University. His current research interests include petri net, discrete event systems, and robust control.

KEYI XING received the B.S. degree in mathematics from Northwest University, Xi'an, China, in 1982, the M.S. degree in applied mathematics from Xidian University, Xi'an, in 1985, and the Ph.D. degree in systems engineering from Xi'an Jiaotong University, Xi'an, in 1994.

He was with Xidian University, in 1985. Since 2004, he has been with Xi'an Jiaotong University, where he is currently a Professor of systems engineering with the State Key Laboratory for Manufacturing Systems Engineering and the Systems Engineering Institute. His current research interests include control and scheduling of automated manufacturing, discrete-event, and hybrid systems.