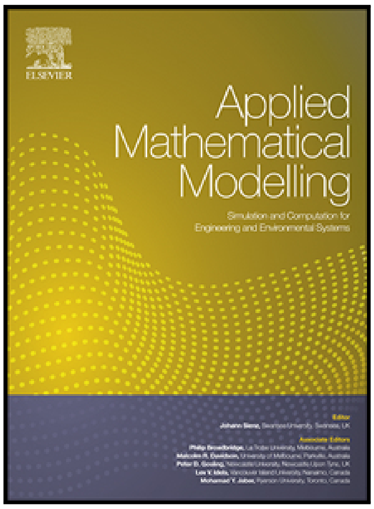
Accepted Manuscript



An estimation of distribution algorithm for scheduling problem of flexible manufacturing systems using Petri nets

XinNian Wang, KeYi Xing, XiaoLing Li, JianChao Luo

PII: S0307-904X(17)30709-6
DOI: 10.1016/j.apm.2017.11.018
Reference: APM 12060

To appear in: *Applied Mathematical Modelling*

Received date: 3 June 2016
Revised date: 10 October 2017
Accepted date: 20 November 2017

Please cite this article as: XinNian Wang, KeYi Xing, XiaoLing Li, JianChao Luo, An estimation of distribution algorithm for scheduling problem of flexible manufacturing systems using Petri nets, *Applied Mathematical Modelling* (2017), doi: 10.1016/j.apm.2017.11.018

Highlights

- This is the first report on applying estimation of distribution algorithm (EDA) to the studied problem.
- A kind of PN-based deadlock controllers for FMSs is imbedded to exclude infeasible individuals.
- An effective voting procedure is adopted to construct the probabilistic model of EDA.
- The longest common subsequence is also embedded in the model for mining excellent genes.
- A new modified variable neighborhood search is developed as an efficiency enhancement of EDA.

An estimation of distribution algorithm for scheduling problem of flexible manufacturing systems using Petri nets

XinNian Wang[1], KeYi Xing[1,*], XiaoLing Li[1], JianChao Luo[1]

Abstract

Based on the place-timed Petri net models of flexible manufacturing systems (FMSs), this paper proposes a novel effective estimation of distribution algorithm (EDA) for solving the scheduling problem of FMSs. A candidate solution is represented as an individual with two sections: the first contains the route information while the second is a permutation with repetition for parts. The feasibility of individuals is checked and guaranteed by a highly permissiveness deadlock controller. A feasible individual is interpreted into a deadlock-free schedule while the infeasible ones are amended. The probabilistic model in EDA is constructed via a voting procedure. An offspring individual is then produced based on the model from a seed individual, and the set of seed individuals is extracted by a roulette method from the current population. The longest common subsequence is also embedded into the probabilistic model for mining good genes. A modified variable neighborhood search is applied on offspring individuals to obtain better solutions in their neighbors and hence to improve EDA's performance. Computational results show that our proposed algorithm outperforms all the existing ones on benchmark examples for the studied problem. It is of important practice significance for the manufacturing of time-critical and multi-type products.

*Corresponding author
Email address: kyxing@mail.xjtu.edu.cn (KeYi Xing)
[1]The State Key Laboratory for Manufacturing Systems Engineering and Systems Engineering Institute, Xi'an Jiaotong University, Xi'an 710049, P. R. China.

Keywords: flexible manufacturing system, timed Petri net, scheduling, estimation of distribution algorithm, variable neighborhood search

1. Introduction

A flexible manufacturing system (FMS) is a computer controlled manufacturing system which consists of a limited set of resources and is capable to process multi-types of parts. Typical applications in real-life include eyeglass productions, industrial stamping systems, and semiconductor manufacturing industries [1–4]. These systems generally exhibit high degrees of resource sharing and route flexibility. The competitions for limited shared resources by concurrent processes of various parts may result in deadlocks, if no proper control or scheduling method is applied. Once a deadlock appears, the whole system or a part of it remains indefinitely blocked and cannot finish the task. Thus, it is of paramount importance to develop effective control and scheduling methods to prevent deadlocks while optimize the system performance.

The deadlock problem has been widely researched from the control viewpoint, and many control methods have been proposed [5–10]. Although the deadlock-free operation in FMS is guaranteed via the above deadlock control methods, the system performance is not considered. Scheduling is an integral part for various types of manufacturing systems [11–15], and the scheduling of FMSs involves not only the handling of deadlocks but the optimization of a certain objective function, and therefore is more difficult than a pure deadlock control problem. There are quite a few works on this area [4, 16–25]. Sethi et al. [16] dealt with the problem of sequencing parts and robot moves in a robotic cell. The cycle time formulas are developed and analyzed for cells producing a single part type using two or three machines, and optimal sequences of robot moves are obtained. Ramaswamy and Joshi [17] proposed a mathematical model for automated manufacturing systems (AMSs) with material handling devices and limited buffers. A Lagrangian relaxation heuristic was used to simplify the model for searching the optimized average flow time. To search for the optimal or near-optimal schedule of the semiconductor test facility, Xiong and Zhou [4] proposed two hybrid heuristic strategies by combining the best-first with the controlled backtracking based on the execution of Petri nets (PNs). Abdallah, Elmaraghy, and Elmekkawy [18] used timed PNs to model FMSs and proposed a scheduling algorithm to minimize the mean flow time. Their algorithm is based on a depth-first strategy and the branch and bound principle together with a siphon truncation technique. Dashora et al. [19] used extended colored timed PN to model the dynamic behavior of the system of simple sequential processes with resources (S^3PR) and presented a deadlock-free scheduling method based on an evolutionary endosymbiotic learning automata algorithm. Xing et al. [20] embedded a deadlock avoidance policy (DAP) into a genetic algorithm and developed a deadlock-free genetic algorithm for AMSs. A one-step look-ahead method is used to guarantee the feasibilities of chromosomes and the deadlock-free schedule is then obtained by amending the infeasible ones. Han et al. [21] proposed a new deadlock-free genetic algorithm with different kinds of crossover and mutation operations. The effects of different deadlock controllers were also discussed and compared. Luo et al. [22] developed new scheduling approaches by combining DAPs and hybrid heuristic searches. Based on a PN reachability graph and minimum processing time matrix, new heuristic and selection functions are designed to guide the search. Baruwa et al. [23] proposed an Anytime Layered Search algorithm based on the reachability analysis of timed colored PN. They combines breadth-first iterative deepening A* with suboptimal breadth-first heuristic search and backtracking. Lei et al. [24] proposed an effective hybrid discrete differential evolution algorithm based on the timed PN models of FMSs, where sequence-dependent setup time is considered. A variable neighborhood search is adopted to improve the solutions' qualities. Han et al. [25] proposed a hybrid particle swarm optimization algorithm by incorporating particle normalization and simulated annealing based local search into the algorithm. A random-key based solution representation is designed to encode the schedule into a particle.

Estimation of distribution algorithm (EDA) [26, 27] is an evolutionary algorithm proposed in recent years and has received increasing attentions of many researchers. 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 [28–31]. However, to the best of our knowledge, no work has been done for the scheduling problem of FMSs using EDA. Can EDA be used to solve this problem and obtain more promising results than existing methods?

This work intends to answer the question by proposing a new EDA. A candidate solution of the scheduling problem is represented as an individual of two sections, route information and operation sequence. The first section contains the route information of parts and the second is a permutation with repetition for all the parts. To exclude infeasible individuals, a kind of PN-based deadlock controllers for FMSs is imbedded. Since the probabilistic model of EDA constitutes the main issue and the performance of the algorithm is closely related to it [32], in this work, an effective voting procedure is adopted to construct the model. The longest common subsequence (LCS) that finds the common elements of two individuals is also embedded in the model for mining excellent genes. Then, an offspring individual is produced based on the model from a seed individual, which is selected from the current population by a roulette method. Furthermore, to achieve a better result, a modified variable neighborhood search (MVNS) is developed as an efficiency enhancement of EDA. The local searches in MVNS are modified to accommodate the PN models of FMSs. To the author's knowledge, only a few research works [20–25] study the scheduling problem considered in this paper. Hence, we test and compare our proposed algorithm with all existing comparable examples and works. Experimental results and comparisons show that the proposed scheduling algorithm outperforms the existing ones, as it provides the best known solutions for 13 of 16 benchmark instances among the five compared approaches. Our proposed scheduling algorithm can be also applied to industrial problems such as flexible manufacturing cells and flexible job-shop production systems.

The rest of the paper is organized as follows. Sect. 2 reviews the PN modeling of FMSs and PN-based deadlock controllers. Sect. 3 develops a scheduling method via PN and EDA, together with the MVNS. The experimental results and comparisons are shown in Sect. 4. Sect. 5 concludes this paper.

2. Petri net modeling of FMSs

This section first briefly reviews the basics of Petri nets (PNs), then the PN model of FMS for scheduling and the deadlock controller. For more details, the readers are referred to [7, 20, 33–35].

2.1. Basics of PNs

A PN is a three-tuple $N = (P, T, F)$, where P is a finite set of places, T is a finite set of transitions, and $F \subseteq P \times T) \cup (T \times P)$ is the set of directed arcs. For a given node $x \in P \cup T$, its preset is defined as ${}^\bullet x = \{y \in P \cup T | (y, x) \in F\}$, and the postset $x^\bullet = \{y \in P \cup T | (x, y) \in F\}$.

Let $Z_0 = \{0, 1, 2, \ldots\}$ and $Z_k = 1, 2, \ldots, k$. A *path* is a string $\alpha = x_1 x_2 \ldots x_k$, where $x_i \in P \cup T$ and $(x_i, x_{i+1}) \in F, i \in Z_{k-1}$. A marking or state of N is a mapping $M : P \to Z_0$. For a given marking M and a place $p \in P$, denote $M(p)$ as the number of tokens in p at M. A PN N with an initial marking M_0, denoted as (N, M_0), is called a marked PN.

For a given transition $t \in T$, if $\forall p \in^\bullet t$, $M(p) > 0$, t is enabled at M, denoted by $M[t >$. An enabled transition t at M can fire, yielding M', denoted by $M[t > M'$, where $M'(p) = M(p) - 1, \forall p \in^\bullet t \setminus t^\bullet$, $M'(p) = M(p) + 1$, $\forall p \in t^\bullet \setminus^\bullet t$, and otherwise, $M(p) = M'(p)$. A sequence of transitions $\alpha = t_1 t_2 \ldots t_k$ is feasible from M if $M_i[t_i > M_{i+1}$, $i \in Z_k$, where $M_1 = M$.

A P-timed PN [34, 35] is defined as a three-tuple $(N, M_0, d) = (P, T, F, M_0, d)$, where $(N, M_0) = (P, T, F, M_0)$ is a marked PN, $d : P \to R^+$ is the delay function, and R^+ is the set of nonnegative real numbers. In a P-timed PN, tokens at a marking M are divided into two classes, available and unavailable. A token in place $p \in P$ becomes *available* if it has stayed in p for at least $d(p)$ time units;

otherwise, it is *unavailable*. For a given marking M and a place $p \in P$, denote $M^a(p)$ and $M^u(p)$ as the number of available and unavailable tokens in p at M, respectively. In P-timed PNs, a transition $t \in T$ is *enabled* at any time instance if $\forall p \in^{\bullet} t$, $M^a(p) > 0$. The *firing* of transition t at M yields a new marking M' by removing one available token from each t's input place, and deposits one token into each t's output place.

2.2. FMSs and their P-Timed PN models

In this paper, the studied FMS contains m types of resources $R = \{r_k, k \in Z_m\}$ and is able to process n types of parts $Q = \{q_i, i \in Z_n\}$. A resource type may be a robot, buffer or machine. The capacity of a resource type r_k is a positive integer, denoted as $C(r_k)$, indicating the maximum number of parts that r_k can simultaneously process.

The lot size of type-q_i parts is $\phi(q_i)$. A processing route of a part $w_j = o_{j1}o_{j2}\ldots o_{jk}\ldots o_{jL_j}$ is an ordered sequence of operations, where o_{jk} is the kth operation in w_j and L_j is the total number of operations for type-q_i part. For each type-q_i parts, let o_{is} and o_{ie} be two fictitious operations that represent the loading and the unloading of type-q_i parts, respectively. Then, route w_j is redefined as $w_j = o_{is}o_{j1}o_{j2}\ldots o_{jL_j}o_{ie}$. A part may have more than one route and can choose the routes when processing. Let $\Omega = \{w_j | 1 \leq j \leq |\Omega|\}$ be the set of all processing routes, and $\Omega_i \subseteq \Omega$ be the route set of type-q_i parts.

In our PN model, a processing route w_j of a type-q_i part is modeled by a path of transitions and places $\alpha_j = p_{is}t_{j1}p_{j1}t_{j2}p_{j2}\ldots t_{jk}p_{jk}\ldots t_{jL_j}p_{jL_j}t_{j(L_j+1)}p_{ie}$, where p_{is} and p_{ie} represent operations o_{is} and o_{ie}, respectively, operation place p_{jk} represents operation o_{jk}, transition t_{jk} represents the start of o_{jk} and the completion of $o_{j(k-1)}$. Then, for type-q_i parts' processing routes, the marked PN model is defined as

$$(N_i, M_{i0}) = (P_i \cup \{p_{is}, p_{ie}\}, T_i, F_i, M_{i0}), i \in Z_h$$

where $P_i = \bigcup_{1\leq j\leq|\Omega_i|} \{p_{j1}, p_{j2}, \ldots, p_{jL_j}\}$, $T_i = \bigcup_{1\leq j\leq|\Omega_i|} \{t_{j1}, t_{j2}, \ldots, t_{j(L_j+1)}\}$, and $F_i = \bigcup_{1\leq j\leq|\Omega_i|} \{(p_{is}, t_{j1}), (t_{j1}, p_{j1}), (p_{j1}, t_{j2}), \ldots, (p_{jL_j}, t_{j(L_j+1)}), (t_{j(L_j+1)},$

$p_{ie})\}$. M_{i0} is the initial marking, $M_{i0}(p) = 0$, $\forall p \in P_i \cup \{p_{ie}\}$, and $M_{i0}(p_{is}) = \phi(q_i)$. In N_i, $\forall t \in T_i, |^\bullet t| = |t^\bullet| = 1$. A place $p \in P_i$ is called a split place if $|p^\bullet| > 1$. At split places, parts can choose their processing routes.

For each resource type r_k, assign a resource place and denoted also by r_k. The initial marking of r_k is $C(r_k)$. Tokens in r_k indicate available type-r_k resources. Let P_R and $R(p)$ denote the set of all resource places and the resource required by operation place p, respectively. Suppose that each operation requires only one resource and any two successive operations require different types of resources. Then, add arcs from $R(p)$ to each transition in $^\bullet p$ denoting the occupying of $R(p)$, and arcs from each transition in $p^\bullet$ to $R(p)$ denoting the releasing of $R(p)$. Let F_R be the set of arcs related with resource places. The marked PN that models the FMS is defined as:

$$(N, M_0) = (P \cup P_s \cup P_f \cup P_R, T, F, M_0)$$

where $P = \bigcup_{i \in Z_h} P_i$, $P_s = \{p_{is} | i \in Z_h\}$, $P_f = \{p_{ie} | i \in Z_h\}$, $T = \bigcup_{i \in Z_h} T_i$, $F = F_Q \cup F_R$, and $F_Q = \cup_{i \in Z_h} F_i$. The initial marking M_0 is defined as $M_0(p_{is}) = \phi(q_i)$, $\forall p_{is} \in P_s$; $M_0(p) = 0$, $\forall p \in P \cup P_f$; and $M_0(r_k) = C(r_k)$, $\forall r_k \in P_R$.

In this work, the P-timed PN (N, M_0, d) is used to describe the processing time needed by an operation. A time delay $d(p)$ is assigned to each operation place p to denote its processing time. Note that $d(p) = 0$, $\forall p \in P_s \cup P_f \cup P_R$. Such PN is called as *Petri Net for Scheduling* (*PNS*) [20].

When all operations of all parts are completed, the system reaches its final marking M_f, where $M_f(p_{ie}) = M_0(p_{is})$, $\forall p_{ie} \in P_f$; $M_f(r_k) = C(r_k)$, $\forall r_k \in P_R$; and $M_f(p) = 0$, $\forall p \in P \cup P_s$. A feasible sequence of transitions α from M_0 is complete if $M_0[\alpha > M_f$. Then, a schedule is a feasible and complete sequence of transitions α in the PNS, and the scheduling problem of FMSs is to find a feasible and complete sequence of transitions α^* so that its makespan is as small as possible.

Example 1: Consider an FMS that contains five types of resources, $r_1 - r_5$, and can process two types of parts, q_1 and q_2. Type-r_1 and r_5 resources are robots with $C(r_1) = C(r_5) = 1$, and type-r_2, r_3, and r_4 resources are machines

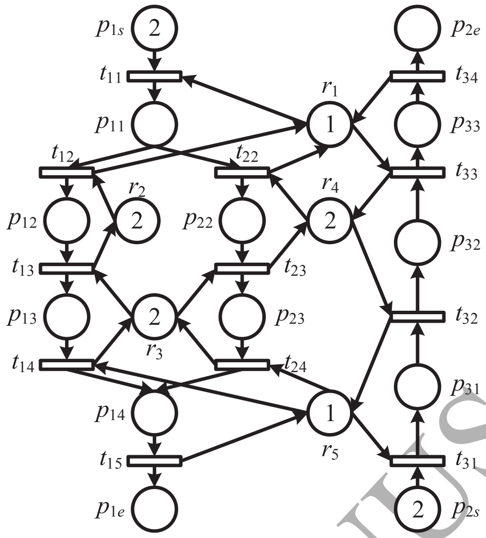


Figure 1: PNS model of the FMS in Example 1.

with $C(r_2) = C(r_3) = C(r_4) = 2$. Type-$q_1$ parts can be processed orderly on r_1, r_2, r_3, and r_5, or on r_1, r_4, r_3, and r_5. Type-q_2 parts are processed orderly on r_5, r_4, and r_1. Thus, type-q_1 parts have two processing routes, $w_1 = p_{1s}t_{11}p_{11}t_{12}p_{12}t_{13}p_{13}t_{14}p_{14}t_{15}p_{1e}$ and $w_2 = p_{1s}t_{11}p_{11}t_{22}p_{22}t_{23}p_{23}t_{24}p_{14}t_{15}p_{1e}$, while type-$q_2$ parts has only one $w_3 = p_{2s}t_{31}p_{31}t_{32}p_{32}t_{33}p_{33}t_{34}p_{2e}$. The PN model of the FMS is shown in Fig. 1, where the required processing parts of type q_1 and q_2 are both 2.

2.3. Deadlock controllers of PNS

In our PNS, a given marking M is a deadlock, if $M \neq M_f$ and no transition is enabled at M. According to the structures of PNSs and S^{3}PRs for FMSs in [5], it can be known that deadlock controllers for S^{3}PR and PNS of the same FMS are the same. Thus, deadlock controllers in [5–9] can directly be used for PNSs. For the purpose of obtaining desirable scheduling results, the deadlock controller in [7] with high permissiveness is used to avoid deadlocks in the proposed scheduling algorithm.

3. Scheduling algorithm based on EDA

In this work, EDA is used for solving the scheduling problem of FMSs with respect to the makespan minimization. DAPs are embedded so that all individuals can be interpreted to feasible schedules. Some constraints of our studied FMSs are described as follows.

1. A resource may be a robot or a machine. The capacity of a resource type is a positive integer and indicates the maximum number of jobs that this resource type can simultaneously handle;
2. A processing route of a part is a predefined sequence of operations. A job may be processed on more than one route and can choose the routes during its processing;
3. Each operation requires one unit resource, and the resource types required for the two successive operations of a job are different;
4. The processing times for operations are prescribed in advance;
5. No pre-emption is allowed.

Estimation of distribution algorithm (EDA) is an evolutionary algorithm that extracts the global statistical information by constructing an explicit probabilistic model from selected solutions [36]. An EDA works with a *population* of candidate *solutions* (*individuals*) to the problem. The population is evaluated by a *fitness function* and the initial population is generated randomly. A set of fitter solutions are selected and a *probabilistic model* that tries to estimate the probability distribution of the selected individuals is then constructed. A new individual is generated by sampling the distribution of the probabilistic model. These new individuals are incorporated back to the old population, replacing it partly or entirely. The process is repeated until some termination criteria are met, with each iteration of this procedure usually referred to as one *generation* of EDA [36].

The various elements of the proposed EDA are detailed as follows.

3.1. Representation, interpretation and reparation

In this paper, permutations with repetitions are used to represent the individuals in EDA. An individual π is rewritten as two sections $\pi = (S_r; S_o)$, where S_r contains the route information for parts and S_o is a permutation with repetition for all the parts. Each part J_i appears $l(J_i)$ times in So, where $l(J_i)$ $= max\,\{l(w_s)|w_s$ is a route of J_i $\}$ and $l(w_s)$ is the length of w_s. In the initial population, both sections, S_r and S_o, are generated randomly for each individual.

Since the ith J_s in S_o represents J_s's ith operation, S_o can be uniquely decoded as a sequence of operations $o(S_o)$, and π can also be rewritten as $\pi = (S_r; o(S_o))$. By associating all the operations to the corresponding transitions, π is interpreted as a sequence of transitions $\alpha(\pi) = t_1t_2 \ldots t_L$, which is regarded as a schedule in PNS. Let O_{ij} be the the jth operation of part J_i and $f(t_k[O_{ij}])$ be the firing time of transition t_k that corresponds to operation O_{ij}. Considering the prescribed operation sequence in PNS and the firing order in $\alpha(\pi)$, $t_k[O_{ij}]$ can be fired only after (1) operation $O_{i(j-1)}$ is finished, and (2) t_{k-1} is fired. Let t_{k-1} and t_s correspond to operations O_{uv} and $O_{i(j-1)}$, respectively. Then, $f(t_k[O_{ij}]) = max\left\{f(t_s[O_{i(j-1)}]) + d(O_{i(j-1)}), f(t_{k-1}[O_{uv}])\right\}$, and the makespan of $\alpha(\pi)$ is

$$\lambda(\alpha(\pi)) = max\,\{f(t_k[O_{ij}]) + d(O_{ij})\} \tag{1}$$

Example 2: Consider the PNS in Fig. 1. There are four parts to be processed: two type-q_1 parts, J_1 and J_2, and two type-q_2 parts, J_3 and J_4. The type-q_1 parts have two routes, w_1 and w_2, while type-q_2 parts have only one, w_3. Assume that J_1 and J_2 are processed on routes w_1 and w_2, respectively. Then, $S_r =$ (w_1, w_2, w_3, w_3) can be used as the first section of an individual π. Since the route lengths $l(J_i) = 5$, 5, 4, and 4, the second section S_o can be represented as a permutation with repetition which contains five J_1's, five J_2's, four J_3's, and four J_4's. For example, $S_o = (J_1, J_1, J_3, J_2, J_3, J_2, J_4, J_1, J_2, J_3, J_4, J_1, J_3, J_4, J_1, J_2,$ $J_2, J_4)$. Then π is represented as $\pi = (S_r; S_o) = (w_1, w_2, w_3, w_3; J_1, J_1, J_3, J_2, J_3,$

$J_2, J_4, J_1, J_2, J_3, J_4, J_1, J_3, J_4, J_1, J_2, J_2, J_4$). On the other hand, S_o can be interpreted as a sequence of operations $o(S_o) = (O_{11}, O_{12}, O_{31}, O_{21}, O_{32}, O_{22}, O_{41}, O_{13}, O_{23}, O_{33}, O_{42}, O_{14}, O_{34}, O_{43}, O_{15}, O_{24}, O_{25}, O_{44})$, and π can be interpreted as a sequence of transitions $\alpha(\pi) = (t_{11}, t_{12}, t_{31}, t_{11}, t_{32}, t_{22}, t_{31}, t_{13}, t_{23}, t_{33}, t_{32}, t_{14}, t_{34}, t_{33}, t_{15}, t_{24}, t_{15}, t_{34})$, or for the details, $\alpha(\pi) = (t_{11}[O_{11}], t_{12}[O_{12}], t_{31}[O_{31}], t_{11}[O_{21}], t_{32}[O_{32}], t_{22}[O_{22}], t_{31}[O_{41}], t_{13}[O_{13}], t_{23}[O_{23}], t_{33}[O_{33}], t_{32}[O_{42}], t_{14}[O_{14}], t_{34}[O_{34}], t_{33}[O_{43}], t_{15}[O_{15}], t_{24}[O_{24}], t_{15}[O_{25}], t_{34}[O_{44}])$, where t_{ij} in $t_{ij}[O_{uv}]$ is the start of operation O_{uv}.

Note that the sequence of transitions generated from an individual by the interpretation aforementioned may be infeasible and lead to deadlocks. Thus, the feasibility of each individual should be checked and the infeasible individuals are translated into feasible ones. In this paper, the amending algorithm proposed in [20] is incorporated to obtain the feasible sequence of transitions from M_0 to M_f. The amending algorithm is based on the deadlock controller proposed in [7]. This deadlock controller is of highly permissiveness and has simple structures. The readers can refer to [7, 20] for more details.

3.2. Probabilistic model

The construction of a probabilistic model is an important procedure that differentiates EDA from other meta-heuristics. This model does not aim to perfectly represent the set of selected individuals but to reveal a general distribution that captures the features of these individuals that make them better than other ones [36]. On the other hand, the efficiencies of the model constructing and information sampling are closely related to the performance of the algorithm. Hence, the choice of the probabilistic model plays a decisive role in EDA's success.

In this paper, a *dominance matrix* D is used as the probabilistic model. Let Π_e denote the elite set that contains the best n_e individuals in the current population. To extract the global statistical information about parts and positions from Π_e and thereby construct the probabilistic model, the voting procedure used in [29, 37] is adopted in this paper.

Given $\pi = (S_r; S_o) \in \Pi_e$. We know that S_o is a permutation with repetition for parts and each part J_k appears $l(J_k)$ times in S_o. To identify the repetitive parts in different positions in S_o, the sequence of operations $o(S_o) = O_1 O_2 \dots O_L$ is used in the voting procedure. Define a reference sequence of operations $\Theta = \theta_1 \theta_2 \dots \theta_L$, which remains unchanged during the procedure. Note that all the elements in $o(S_o)$ are different from each other and an operation O_j in $o(S_o)$ also appears in Θ. Thus, we can assign a unique index i to O_j if $O_j = \theta_i, i, j \in Z_L$. Let $\delta_\pi(i,j)$ be the indicator function for π, where $\delta_\pi(i,j) = 1$ if $O_j = \theta_i$ or O_j has index i; otherwise $\delta_\pi(i,j) = 0$.

Now an $L \times L$ dominance matrix D can be constructed and its (i,j)-entry D_{ij}, which denotes the times (weighted) that operation $\theta_i \in \Theta$ appears at the jth positions in $o(S_o)$s of all the individuals in Π_e, is formally defined as follows.

$$D_{ij} = \sum_{\pi \in \Pi_e} \delta_\pi(i,j) \times (\lambda(\alpha(\pi_w)) - \lambda(\alpha(\pi)) + 1)/(\lambda(\alpha(\pi_w)) - \lambda(\alpha(\pi_b)) + 1) \quad (2)$$

where π_b and π_w are the best and worst individuals in the current population, respectively.

Example 3: Consider the PNS in Fig. 1. For simplicity, the required processing parts of type q_1 and q_2 are both set to 1. The processing time of operations is randomly taken, with $d(p_{11}) = 4, d(p_{12}) = 32, d(p_{13}) = 38, d(p_{14}) = 5, d(p_{22}) = 23, d(p_{23}) = 20, d(p_{31}) = 5, d(p_{32}) = 22, d(p_{33}) = 6$. Given a population of 5 individuals:

$\pi_1 = (w_2, w_3; J_1, J_2, J_2, J_1, J_1, J_2, J_1, J_2, J_1), \lambda(\alpha(\pi_1)) = 53;$

$\pi_2 = (w_2, w_3; J_1, J_2, J_2, J_1, J_2, J_2, J_1, J_1, J_1), \lambda(\alpha(\pi_2)) = 58;$

$\pi_3 = (w_2, w_3; J_1, J_2, J_1, J_1, J_2, J_2, J_2, J_1, J_1), \lambda(\alpha(\pi_3)) = 60;$

$\pi_4 = (w_1, w_3; J_1, J_1, J_2, J_1, J_2, J_1, J_1, J_2, J_2), \lambda(\alpha(\pi_4)) = 85;$

$\pi_5 = (w_1, w_3; J_1, J_1, J_1, J_1, J_1, J_2, J_2, J_2, J_2), \lambda(\alpha(\pi_5)) = 112;$

Assume that the elite set Π_e contains 4 individuals, $\pi_1 - \pi_4$. It can be known that the sequence of operations corresponding to $\pi_2 \in \Pi_e$ is $o(S_{o2}) = (O_{11}, O_{21}, O_{22}, O_{12}, O_{23}, O_{24}, O_{13}, O_{14}, O_{15})$. Then, set the reference sequence of operations as $\Theta = (\mathrm{O}_{11}, \mathrm{O}_{12}, \mathrm{O}_{13}, \mathrm{O}_{14}, \mathrm{O}_{15}, \mathrm{O}_{21}, \mathrm{O}_{22}, \mathrm{O}_{23}, \mathrm{O}_{24})$, and $\delta_{\pi_1}(i,j)$ can be calculated. For example, $\delta_{\pi_1}(3,7) = 1$ since the 3rd operation O_{13} in Θ

appears at the 7th position in $o(S_{o2})$. Note that for π_2, its weight $(\lambda(\alpha(\pi_w)) - \lambda(\alpha(\pi_2)+1)/\ (\lambda(\alpha(\pi_w)) - \lambda(\alpha(\pi_b))+1) = 0.92$, hence, π_2's votes for D_{37} is also 0.92 and the value of D_{37} is increased by 0.92. After all the individuals in the elite set Π_e have voted, the final dominance matrix is obtained as follows:

$$\begin{bmatrix} 3.27 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.47 & 0.88 & 1.92 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.35 & 1 & 0 & 0.92 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.47 & 1 & 1.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.47 & 0 & 2.8 \\ 0 & 2.8 & 0.47 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.92 & 0 & 1.35 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.92 & 1.88 & 0 & 0.47 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.92 & 0.88 & 1 & 0.47 \end{bmatrix}$$

Inspired by [29], the longest common subsequence (LCS) is also embedded in the model to find good genes of two individuals.

A subsequence of a given sequence X is the sequence X with zero or more elements deleted. Given two sequences X and Y, Z is their common subsequence if Z is a subsequence of both X and Y. Let Γ_{XY} denote the set of all common subsequences of X and Y. Then, the longest one in Γ_{XY} is called their longest common subsequence (LCS) [38].

The brute-force procedure to solve the LCS problem is obviously of exponential complexity. A basic idea to simplify the procedure is using *dynamic programming*, which reduces its complexity to $O(L_a L_b)$, where L_a and L_b are the length of two given sequences, respectively. The readers can refer to [38] for details. Note that we also use the sequence of operations in the procedure to identify the repetitive parts.

3.3. Offspring reproduction and replacement

In our EDA, each offspring is reproduced from the seed, while the set of seeds, denoted by Π_s ($|\Pi_s| = n_s$), is extracted by the roulette method from the current population. Let $\pi_b = (S_{rb}, S_{ob})$ be the best individual found so far, and $\pi = (S_r, S_o) \in \Pi_s$ be a seed. The LCS of $o(S_o)$ and $o(S_{ob})$ is considered as good genes, denoted as σ_L. Let $\pi_f = (S_{rf}, S_{of})$ be the offspring individual reproduced from π. It is reproduced as follows.

First, if $O_{ij} \in \sigma_L$, $t_k[O_{ij}]$ in $\alpha(\pi)$ is scheduled to the kth position in $\alpha(\pi_f)$ with a probability p_l that depends on the length of σ_L. In this work, we defines $p_l = exp(\mu L(\sigma_L)/L_o)$, where $\mu = log(0.8)$ is a constant, $L(\sigma_L)$ is the length of σ_L, and L_o is the length of the sequence of operations. Let Ψ_u be the set of unscheduled transitions in $\alpha(\pi)$.

Then, for each unassigned qth position in $\alpha(\pi_f)$, select a transition $t_p[O_{uv}]$ randomly from Ψ_u with probability $D_{pq}/\sum_{t_k \in \Psi_u} D_{kq}$, where D_{ij} is the (i,j)-entry of D. At last, set the route information in S_{rf} and the parts in S_{of} accordingly. When an offspring individual is reproduced, if it is better than the worst one and different from the others in the current population, replace the worst one with it. This replacement procedure apparently keeps the population diversity while improving the population quality.

3.4. Local search

To further improve the EDAs performance and prevent it from being stuck in local optima, a local search method, variable neighbourhood search (VNS), is employed. Once a new individual π_f is produced, VNS is applied on it with a probability p_v that depends on the quality of π_f. This work defines $p_v = min\{max\{exp(H/\gamma), \epsilon\}, 1\}$, where γ and ϵ are constants with $\gamma = 0.2/log(0.02)$ and $\epsilon = 0.01$, $H = (\lambda(\alpha(\pi_f)) - \lambda(\alpha(\pi_b)))/\lambda(\alpha(\pi_b))$, and π_b is the best solution found so far.

In this paper, two neighborhood structures, *swap_local _search* and *insert_local_search* [28], are used in VNS. For a given individual, the former leads to some possible swapping pairs of parts, while the latter consists of some operations of inserting one part in front of another part. Since they both change the sequence of processing parts, the individuals generated by them may not be feasible. Amending infeasible individuals into feasible ones is a very timeconsuming procedure. Thus, to improve the search efficiency and ensure the feasibility of generated individuals at the same time, this work develops two new local searches based on the following Remarks.

Remark 1: Let $\pi = (S_r; S_o)$ be a feasible individual and $\alpha(\pi)$ be the corresponding sequence of transitions. Each part J_i appears $l(J_i)$ times in S_o and each transition $t_{jk} \in T$ appears $g(w_s)$ times in $\alpha(\pi)$, where $g(w_s)$ is the number of parts processed by route w_s. Then, the swapping pairs of the same part J_i at different positions in S_o, or the same transition t_{jk} at different positions in $\alpha(\pi)$ do not change $\pi = (S_r; S_o)$ and $\alpha(\pi)$.

Remark 2: Let $\pi = (S_r; S_o)$ be a feasible individual and $\alpha(\pi) = t_1 t_2 \ldots t_n$. Let ${}^{(p)}t$ and $t^{(p)}$ denote the sets of input and output operation places of transition t, respectively. For a transition t_k in $\alpha(\pi)$, let $a_\pi(t_k)$ and $b_\pi(t_k)$ denote the minimum and maximum subscripts of transitions in ${}^\bullet({}^{(p)}t_k)$ and $(t_k^{(p)})^\bullet$, respectively. Then, t_k cannot fire before the firing of the transition with subscript $a_\pi(t_k)$ or after the firing of the transition with subscript $b_\pi(t_k)$ in $\alpha(\pi)$.

In *swap_local_search*, a pair of transitions $t_i[O_{pq}]$ and $t_j[O_{uv}]$ to be swapped must satisfy two constraints: $J_p \neq J_u$, $t_i \neq t_j$ and $a_\pi(t_j[O_{uv}]) < i < b_\pi(t_j[O_{uv}])$, $a_\pi(t_i[O_{pq}]) < j < b_\pi(t_i[O_{pq}])$. The search stops if no better local optimum is found. The first new local search *deadlock-free swap_local _search* (DSLS) is stated as follows.

Example 4: Consider the PNS in Fig. 1. Let $\pi = (S_r; S_o) = (w_1, w_2, w_3; J_1, J_1, J_3, J_2, J_3, J_3, J_1, J_2, J_1, J_2, J_2, J_1, J_2, J_3)$ be an individual that is inputted to algorithm DSLS. Then $\alpha(\pi) = (t_{11}, t_{12}, t_{31}, t_{11}, t_{32}, t_{33}, t_{13}, t_{22}, t_{14}, t_{23}, t_{24}, t_{15}, t_{15}, t_{34})$. At the beginning of DSLS, $i = 1$ and $j = i+1$. $J_i = J_j = J_1$. No operation swap $\Lambda(\pi; i, j)$ is done.

For $i = 9$, $j = 13$, suppose that no better individual has been obtained, i.e., the swap moves will still be performed on π, with respect to J_1 in the ith position and J_2 in the jth position. Since $a_\pi(t_i) = 7$ and $b_\pi(t_i) = 13$, this swap move is unpermitted under the second condition of DSLS. In fact, this move will turn the individual into $\pi' = (w_1, w_2, w_3; J_1, J_1, J_3, J_2, J_3, J_3, J_1, J_2, \boldsymbol{J_2}, J_2, J_2, J_1, \boldsymbol{J_1}, J_3)$ (differences are shown in bold), which is not feasible and must be amended. Af-

Algorithm 1 DSLS

Input: a feasible individual π_c and $\alpha(\pi_c)$;

set $\pi_l = \pi_c$; $\alpha(\pi_l) = \alpha(\pi_c)$; $i = 1$; // π_l is the best individual found in DSLS

while termination criterion is not satisfied **do**

  $j = i + 1$; compute $a_{\pi_l}(t_i[O_{pq}])$, $b_{\pi_l}(t_i[O_{pq}])$, $a_{\pi_l}(t_j[O_{uv}])$, and $b_{\pi_l}(t_j[O_{uv}])$; // t_i is the ith transition in $\alpha(\pi_l)$

  while $a_{\pi_l}(t_i[O_{pq}]) < j < b_{\pi_l}(t_i[O_{pq}])$ **do**

    if $J_p \neq J_u$, $t_i \neq t_j$, and $a_{\pi_l}(t_j[O_{uv}]) < i < b_{\pi_l}(t_j[O_{uv}])$ **then**

      $\pi_t = \Lambda(\pi_l; i, j)$; // Λ permutes the parts in positions i and j in π_l

      amend π_t and $\alpha(\pi_t)$;

      if $\lambda(\alpha(\pi_t) < \lambda(\alpha(\pi_l)$ **then**

        $\pi_l = \pi_t$; $\alpha(\pi_l) = \alpha(\pi_t)$; $i = i - 1$; $j = i + 1$;

      else

        $j = j + 1$;

      end if

    else

      $j = j + 1$;

    end if

  end while

  $i = i + 1$;

  if $i > N - 1$ **then**

    $i = 1$;

  end if

end while

Output: π_l and $\alpha(\pi_l)$;

ter that, we have the amended individual $\pi'' = (w_1, w_2, w_3; J_1, J_1, J_3, J_2, J_3, J_3, J_1, J_2, J_1, J_2, J_2, \boldsymbol{J}_2, \boldsymbol{J}_1, J_3)$ and its corresponding sequence of transitions is the same as $\alpha(\pi) = (t_{11}, t_{12}, t_{31}, t_{11}, t_{32}, t_{33}, t_{13}, t_{22}, t_{14}, t_{23}, t_{24}, t_{15}, t_{15}, t_{34})$. Such situations are apparently invalid and hence must be prevented by DSLS.

In *insert_local_search*, for the transition $t_i[O_{pq}]$ that is to be inserted to the front of $t_j[O_{uv}]$ in $\alpha(\pi)$, only one constraint is required: $a_\pi(t_i[O_{pq}]) < j < b_\pi(t_i[O_{pq}])$. The search stops if no better local optimum is found. The second new local search *deadlock-free insert_local_search* (DILS) can be established as follows.

Algorithm 2 DILS

Input: a feasible individual π_c and $\alpha(\pi_c)$;
  set $\pi_l = \pi_c$; $\alpha(\pi_l) = \alpha(\pi_c)$; $i = 1$; // π_l is the best individual found in DILS
  while termination criterion is not satisfied **do**
    $j = 1$; compute $a_{\pi_l}(t_i[O_{pq}])$ and $b_{\pi_l}(t_i[O_{pq}])$; // t_i is the ith transition in $\alpha(\pi_l)$
    while $a_{\pi_l}(t_i[O_{pq}]) < j < b_{\pi_l}(t_i[O_{pq}])$ **do**
      $\pi_t = \Delta(\pi_l; i, j)$; // operator Δ inserts the ith part to position j in π_l
      amend π_t and $\alpha(\pi_t)$;
      if $\lambda(\alpha(\pi_t) < \lambda(\alpha(\pi_l)$ **then**
        $\pi_l = \pi_t$; $\alpha(\pi_l) = \alpha(\pi_t)$; $i = i - 1$; $j = 1$;
      else
        $j = j + 1$;
      end if
    end while
    $i = i + 1$;
    if $i > N$ **then**
      $i = 1$;
    end if
  end while
Output: π_l and $\alpha(\pi_l)$;

According to these two new local searches, the swap and insert operations are performed within proper domains that are determined by the characteristics

(*Remarks* 1 and 2) of the concerned system. Thus, a number of invalid operations are avoided and the local search efficiency is therefore improved. In the following, a new Modified VNS (MVNS) is developed based on DSLS and DILS.

Algorithm 3 MVNS

Input: an offspring individual π_f and $\alpha(\pi_f)$;
  set $\pi_v = \pi_c = \pi_f$; $\alpha(\pi_v) = \alpha(\pi_c) = \alpha(\pi_f)$; // π_v is the best individual found in MVNS
  while termination criterion is not satisfied **do**
    repeat
      $\lambda_1 = \lambda(\alpha(\pi_c))$; $(\pi_d, \alpha(\pi_d)) = \text{DSLS}(\pi_c, \alpha(\pi_c))$;
      $(\pi_c, \alpha(\pi_c)) = \text{DILS}(\pi_d, \alpha(\pi_d))$; $\lambda_2 = \lambda(\alpha(\pi_d))$;
    until $\lambda_1 \geq \lambda_2$
    if $\lambda(\alpha(\pi_c)) < \lambda(\alpha(\pi_v))$ **then**
      $\pi_v = \pi_c, \alpha(\pi_v) = \alpha(\pi_c)$;
    end if
    choose two different positions randomly in π_c and permute the parts in these two positions;
    amend π_c and $\alpha(\pi_c)$;
  end while
Output: π_v and $\alpha(\pi_v)$;

In MVNS, DSLS and DILS are used alternately starting from π_f until no better solution is found. If the so-obtained solution is better than the best solution π_v found so far, the former replaces the latter and the number of iterations is reset to 0. Otherwise, the number of iterations increases by one. Next, choose two different parts in π_v randomly and permute them. The whole procedure is repeated until the maximal number of iterations I_m is reached. Here, the deadlock controller is embedded such that the solutions obtained by MVNS are feasible.

3.5. The hybrid scheduling algorithm

By embedding MVNS and the deadlock controller into EDA, a novel hybrid scheduling algorithm DEDA_MVNS is developed. The algorithm terminates when a given time for computation is reached. Let K denote the population size, the proposed scheduling algorithm is described as follows.

Algorithm 4 DEDA_MVNS

Input: parameters K, n_e, n_s;
  generate an initial population Π_c with K individuals randomly;
  select the best n_e individuals as Π_e; let π_b be the best individual in Π_c;
  while termination criteria are not met **do**
    construct D from Π_e;
    let Π_s be the set of n_s individuals selected by roulette method from Π_c;
    for each $\pi \in \Pi_s$ **do**
      construct LCS of π and π_b;
      reproduce offspring individual π_f based on σ_L and D;
      perform MVNS on π_f with probability p_v;
      if $\lambda(\alpha(\pi_f)) < \lambda(\alpha(\pi_w))$ and π_f does not exist in Π_c **then**
        replace π_w with π_f; // π_w is the worst individual in Π_c;
      end if
    end for
    update Π_e and π_b;
  end while
Output: π_b and $\alpha(\pi_b)$;

---

4. Computational results

In this paper, a widely researched FMS [5–9, 20–25] is used to test the performance of the proposed algorithms. As seen in Fig. 2, this FMS contains 4 machines m_1-m_4 and 3 robots r_1-r_3, and is able to process 3 types of parts J_1-J_3. Its PNS is shown in Fig. 3. The processing time of parts is taken from [20] and shown in Table 1. 16 benchmark instances from [21] based on the PN

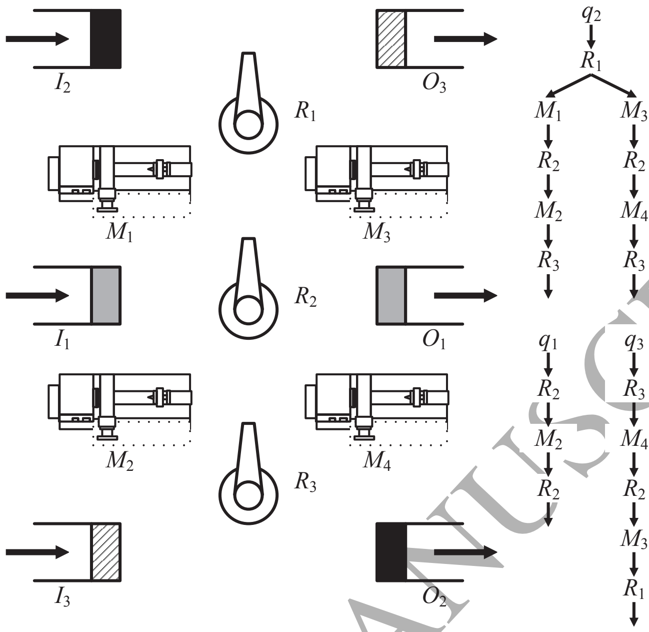


Figure 2: An example of FMS

model of this FMS are tested. According to the resource capacities ($C(m_i)$, $i = 1, 2, 3, 4$, and $C(r_i)$, $i = 1, 2, 3$), these 16 instances are divided into 4 groups. Each group contains 4 instances with the numbers of type-J_1, J_2, and J_3 parts (8, 12, 8), (10, 20, 10), (15, 20, 15), and (20, 20, 20), respectively.

1. In01-In04: $C(m_i) = 2$ and $C(r_i) = 1$;
2. In01-In04: $C(m_i) = 2$ and $C(r_i) = 2$;
3. In09-In12: $C(m_i) = 3$ and $C(r_i) = 2$;
4. In13-In16: $C(m_i) = 3$ and $C(r_i) = 3$.

In the following, we first test the proposed hybrid algorithm and stand-alone ones, then compare the hybrid one with other existing works. All the algorithms are coded in C++ and run on a desktop PC with 3.2 GHz processor and 8 GB memory.

Table 1: Processing time for the FMS in Fig. 2

Type-J_1	Type-J_2		Type-J_3
w_1	w_2	w_3	w_4
$O_{11} : 8$	$O_{21} : 4$	$O_{21} : 4$	$O_{41} : 5$
$O_{12} : 34$	$O_{22} : 32$	$O_{32} : 23$	$O_{42} : 22$
$O_{13} : 5$	$O_{23} : 8$	$O_{33} : 6$	$O_{43} : 4$
	$O_{24} : 38$	$O_{34} : 20$	$O_{44} : 17$
	$O_{25} : 5$	$O_{25} : 5$	$O_{45} : 6$

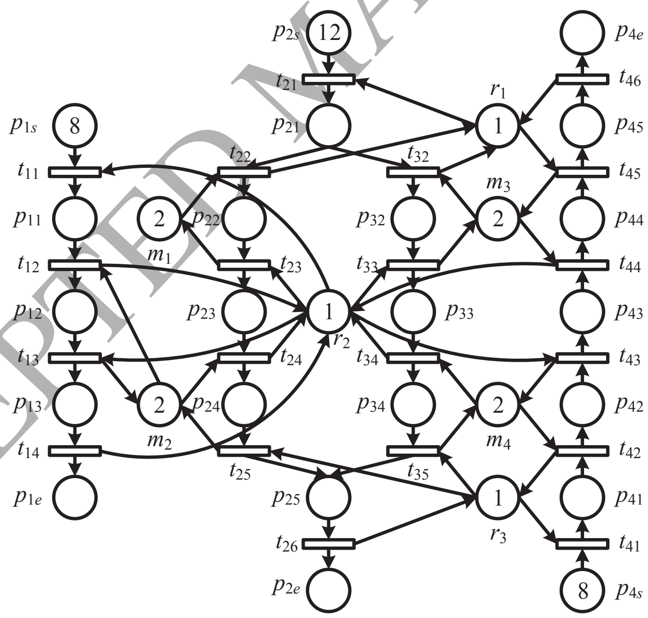


Figure 3: PNS of the FMS in Fig. 2

Table 2: Scheduling results of hybrid algorithms and stand-alone ones

	DEDA		VNS		MVNS		DEDA_VNS		DEDA_MVNS	
Instance	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
In01	349	385.8	298	307.45	296	303.45	287	297.6	**280**	**287.45**
In02	490	554.5	434	450.0	422	436.55	398	411.05	**377**	**397.35**
In03	621	691.85	549	586.95	533	568.6	488	522.3	**483**	**517.3**
In04	772	852.9	663	705.9	653	692.95	609	634.4	**587**	**619.9**
In05	299	321.0	252	261.75	249	258.2	**224**	233.75	225	**231.85**
In06	411	470.25	370	392.85	358	376.15	340	355.65	**321**	**332.3**
In07	510	579.3	442	476.5	429	458.05	412	429.3	**396**	**412.7**
In08	643	720.65	558	584.65	544	570.2	517	538.85	**494**	**515.65**
In09	247	275.4	197	208.6	198	206.5	174	185.0	**168**	**178.3**
In10	340	389.45	286	307.25	287	306.15	252	260.8	**242**	**252.95**
In11	349	385.8	298	307.45	296	303.45	287	297.6	**280**	**287.45**
In12	452	521.05	356	375.2	348	365.35	323	337.75	**309**	**319.0**
In13	232	256.2	183	194.8	184	193.2	**163**	172.2	**163**	**169.1**
In14	319	355.3	259	280.75	250	269.75	242	250.95	**224**	**233.55**
In15	424	478.45	332	358.35	319	338.8	289	308.6	**276**	**291.7**
In16	523	584.0	411	426.2	399	425.5	358	377.8	**346**	**361.8**

4.1. Scheduling results of hybrid algorithms and stand-alone ones

To demonstrate the effectiveness of our proposed algorithms, DEDA, VNS, MVNS, DEDA_VNS, and DEDA_MVNS are tested and compared. The population sizes for instances with 28, 40, 50, and 60 parts are K = 100, 200, 200, and 300, respectively. For each instance, the elite and seed sets both have sizes $0.1 \times K$, i.e., $n_e = n_s = 0.1 \times K$. The maximal iteration number I_m for VNS or MVNS is set as 50. To guarantee the comparisons fairness, the termination criterion of different algorithms are all set as a maximum execution time of $40 \times N_j$ seconds, for the same instance, where N_j is the number of total parts. The simulation results are shown in Table 2, where Best and Average are the best and average makespans among 20 random trials, respectively.

The results show that the hybrid algorithms DEDA_VNS and DEDA_MVNS outperform the stand-alone ones, DEDA, VNS, and MVNS, for all 16 instances. Most significantly improved results of the hybrid algorithms appear in instances with larger lot sizes. The effects of the modification on the local search methods can be demonstrated by the comparison between the results of VNS and MVNS. According to Table 2, MVNS finds better results for 13 instances out of 16, while VNS is preferable for only 3 instances. The similar situation also occurs

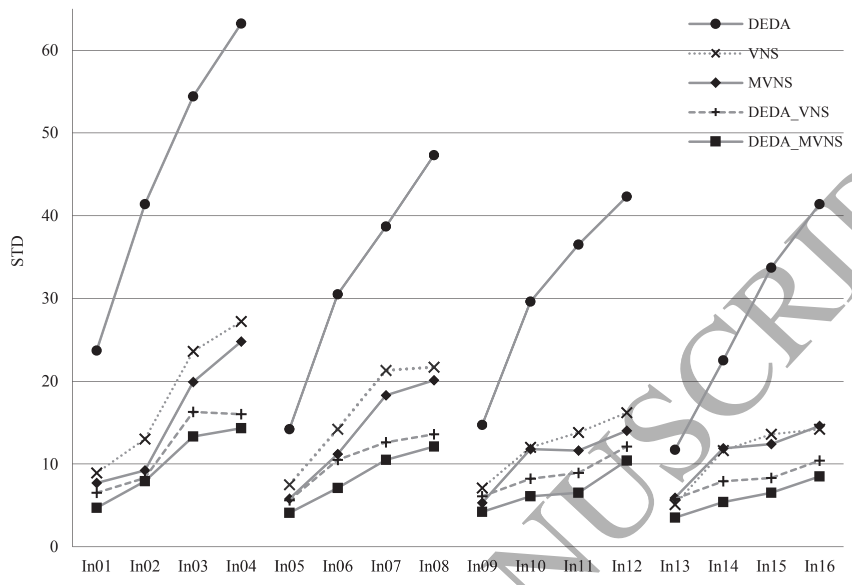


Figure 4: Standard deviations of scheduling results in Table 2.

in the comparison among the hybrid algorithms. DEDA_MVNS provides the best results for 15 instances out of 16, while DEDA_VNS provides only 2. These improvements may be attributed to our modified MVNS that constrain the swap and insert operations in the proper domains to improve the search efficiencies.

The standard deviations of five algorithms for these 16 instances are shown in Fig. 4. It can be seen that DEDA_VNS and DEDA_MVNS significantly outperform DEDA, VNS, and MVNS with respect to the standard deviation. This means the stand-alone algorithms become more robust if combining with others.

4.2. Comparing with other works

In this subsection, the proposed DEDA_MVNS is compared with four existing approaches, D^{2}WS [22], ALS [23], DDE_VNS [24], and HPSO [25], for the 16 instances in Sect. 4.1. The computational results of D^{2}WS, ALS, DDE_VNS, and HPSO are directly from the respective papers. The computational results

Table 3: Scheduling results of DEDA_MVNS and existing approaches.

	DEDA_MVNS		HPSO		DDE_VNS		ALS		D^2WS	
Instance	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
In01	**280**	**287.45**	283	291.5	288	295.5	293	/	303	334.2
In02	**377**	**397.35**	384	406.65	389	403.4	397	/	426	454.9
In03	**483**	**517.3**	485	525.05	**483**	521.55	490	/	539	581.1
In04	**587**	**619.9**	617	650.7	605	639.05	**587**	/	672	739.7
In05	225	231.85	**224**	**231.5**	226	232.7	266	/	252	272.6
In06	**321**	**332.3**	323	333.7	365	379.2	368	/	360	388.6
In07	**396**	**412.7**	404	420.9	416	431.75	450	/	438	483.1
In08	**494**	**515.65**	499	523.25	548	566.05	569	/	569	593.5
In09	**168**	178.3	**168**	**177.6**	188	192.1	196	/	194	207.6
In10	**242**	**252.95**	245	254.15	252	261.4	269	/	269	286.5
In11	**309**	**319.0**	313	323.4	346	365.45	332	/	343	367.1
In12	**373**	**398.05**	380	401.1	394	412.5	409	/	403	440.2
In13	163	169.1	**158**	**162.2**	**158**	**162.2**	186	/	182	193.0
In14	224	233.55	**221**	**232.0**	251	267.5	275	/	246	261.5
In15	**276**	**291.7**	281	295.25	287	308.25	327	/	305	330.6
In16	**346**	**361.8**	**346**	363.4	355	382.15	396	/	359	385.5

are summarized in Table 3. Note that only the best makespans of ALS are provided.

As seen in Table 3, the proposed DEDA_MVNS provides the best solutions for 13 of 16 instances among the five approaches while HPSO, DDE_VNS, and ALS provide 5, 2, and 1, respectively. Moreover, DEDA_MVNS outperforms D^2WS for all the instances. It appears that DEDA_MVNS outperforms these existing approaches, especially for the instances with larger lot sizes and stricter resource constraints.

A non-parametric Friedman's test is also applied to the BSTs of the five different approaches (DEDA_MVNS, HPSO, DDE_VNS, ALS and D^2WS) to clarify whether there are statistical differences among the results. A level $\alpha = 0.05$ of significance is set. The statistical test shows that significant differences exist among the results of these approaches. Therefore, a multiple comparison proposed by [39] is carried out to decide which approach is better. The significance level is also set to $\alpha = 0.05$. The comparisons show that DEDA_MVNS and HPSO outperforms ALS and D^2WS significantly while DEDA_MVNS outperforms DDE_VNS. No statistical differences are found between other pairs.

The computational cost comparison is shown in Table 4. Note that only

Table 4: Execution time of DEDA_MVNS, HPSO and ALS.

Instance	Execution time (seconds)		
	DEDA_MVNS	HPSO	ALS
In01	**1120**	1400	2436
In02	**1600**	2000	1637
In03	2000	2500	**587**
In04	**2400**	3000	3058
In05	1120	1400	**403**
In06	1600	2000	**556**
In07	2000	2500	**853**
In08	2400	3000	**292**
In09	1120	1400	**576**
In10	**1600**	2000	1931
In11	2000	2500	**1024**
In12	2400	3000	**1889**
In13	**1120**	1400	3237
In14	**1600**	2000	1742
In15	2000	2500	**1399**
In16	2400	3000	**1353**

the shortest (178 seconds for In13) and longest (1900 seconds for In 16) execution time of D^2WS were specifically given, and the execution time of DDE_VNS were not reported in [24]. Hence, only the execution time (in seconds) of three scheduling approaches, DEDA_MVNS, HPSO and ALS, are listed in Table 4. Note that DEDA_MVNS and HPSO terminate when some given maximum execution time reaches, while ALS terminates when its two upper bounds converge or some given maximum execution time reaches [23, 25]. The comparison shows that ALS and DEDA_MVNS cost the least time for 10 and 6 instances out of 16, respectively. Non-parametric Friedman's test ($\alpha = 0.05$) shows that significant differences exist among the execution time of three approaches. Multiple comparisons [39] ($\alpha = 0.05$) show that DEDA_MVNS and ALS cost less time than HPSO significantly while no statistical difference is found between DEDA_MVNS and ALS. Additionally, the execution time of D^2WS in most instances are less than 800 seconds [22]. Thus, it can be concluded that D^2WS performs the best in the computational efficiency, followed by two algorithms at the same level, ALS and DEDA_MVNS, while HPSO performs the worst among the four.

5. Conclusions

In this paper, a novel scheduling algorithm DEDA_MVNS using PNs and EDA is developed for the FMSs. Permutations with repetitions for parts are used to represent the individuals and the feasibility of each individual is guaranteed by a highly permissiveness deadlock controller. The probabilistic model in DEDA_MVNS is constructed by the voting procedure and the LCS is incorporated in the model for mining excellent genes. The LCS of the sequences of operations of the seed individual that is selected from the current population by the roulette method, and the best individual found so far, are treated as excellent genes. A local search method MVNS, in which the domains of the swap and insert operations are constrained, is also introduced as an efficiency enhancement for improving the performance.

Experimental results show that the proposed scheduling algorithm outperforms all the existing ones on the benchmark examples. It may be owing to the use of the probability distribution model that can mine desired genes, and the modified local search method MVNS that prevents the algorithm from trapping into local optimum. More studies about the practical effects of these two procedures are required in the future.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grants 61573278.

References

[1] Narciso ME, Piera MA, Guasch A, A time stamp reduction method for state space exploration using colored Petri nets. Simulation 88(5) (2012) 592-616

[2] Pang CK, Cao VL, Optimization of total energy consumption in flexible manufacturing systems using weighted p-timed Petri nets and dynamic programming. IEEE T Autom Sci Eng 11(4) (2014) 1083-1096

[3] Li XL, Xing KY, Wu YC, Wang XN, Luo JC, Total Energy Consumption Optimization via Genetic Algorithm in Flexible Manufacturing Systems. Comput Ind Eng 104 (2017) 188-200

[4] Xiong HH, Zhou MC, Scheduling of Semiconductor Test Facility via Petri Nets and Hybrid Heuristic Search. IEEE T Semiconduct M 11(3) (1998) 384-393

[5] Ezpeleta J, Colom J, Martinez J A, Petri-Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems. IEEE T Robot Autom 11(2) (1995) 173-184

[6] Huang YS, Jeng MD, Xie XL, Chung SL, Deadlock prevention policy based on Petri nets and siphons. Int J Prod Res 39(2) (2001) 283-305

[7] Piroddi L, Cordone R, Furnagalli I, Selective Siphon Control for Deadlock Prevention in Petri Nets. IEEE T Syst Man Cy A 38(6) (2008) 1337-1348

[8] Xing KY, Zhou MC, Liu HX, Tian F, Optimal Petri-Net-Based Polynomial-Complexity Deadlock-Avoidance Policies for Automated Manufacturing Systems. IEEE T Syst Man Cy A 39(1) (2009) 188-199

[9] Liu HX, Xing KY, Zhou MC, Han LB, Wang F, Transition Cover-Based Design of Petri Net Controllers for Automated Manufacturing Systems. IEEE T Syst Man Cy-S 44(2) (2014) 196-208

[10] Feng YX, Xing KY, Gao ZX, Wu YC, Transition Cover-Based Robust Petri Net Controllers for Automated Manufacturing Systems With a Type of Unreliable Resources. IEEE T Syst Man Cy-S 10.1109/TSMC.2016.2558106

[11] Sakhaii M, Tavakkoli-Moghaddam R, Bagheri M, Vatani B, A robust optimization approach for an integrated dynamic cellular manufacturing system

and production planning with unreliable machines. Appl Math Model 40(1) (2016) 169-191

[12] Azadeh A, Ravanbakhsh M, Rezaei-Malek M, Sheikhalishahi M, Taheri-Moghaddam A, Unique NSGA-II and MOPSO algorithms for improved dynamic cellular manufacturing systems considering human factors. Appl Math Model 48 (2017) 655-672

[13] Allahverdi A, Aydilek H, Aydilek A, Two-stage assembly scheduling problem for minimizing total tardiness with setup times. Appl Math Model 40(17-18) (2016) 7796-7815

[14] Karimi S, Ardalan Z, Naderi B, Mohammadi M, Scheduling flexible job-shops with transportation times: Mathematical models and a hybrid imperialist competitive algorithm. Appl Math Model 41 (2016) 667-682

[15] Ham A, Flexible job shop scheduling problem for parallel batch processing machine with compatible job families. Appl Math Model 45 (2017) 551-562

[16] Sethi SP, Sriskandarajah C, Sorger G, Blazewicz J, Kubiak W, Sequencing of parts and robot moves in a robotic cell. Int J Flex Manuf Sys 4(3) (1992) 331-358.

[17] Ramaswamy SE, Joshi SB, Deadlock-free schedules for automated manufacturing workstations. IEEE T Robot Autom 12(3) (1996) 391-400

[18] Abdallah B, Elmaraghy HA, Elmekkawy T, Deadlock-free Scheduling in Flexible Manufacturing Systems Using Petri Nets. Int J Prod Res 40(12) (2002) 2733-2756

[19] Dashora Y, Kumar S, Tiwari MK, Newman ST, Deadlock-free scheduling of an automated manufacturing system using an enhanced colored time resource Petri-net model-based Evolutionary Endosymbiotic Learning Automata approach. Int J Flex Manuf Sys 19(4) (2007) 486-515

[20] Xing KY, Han LB, Zhou MC, Wang F, Deadlock-Free Genetic Scheduling Algorithm for Automated Manufacturing Systems Based on Deadlock Control Policy. IEEE T Syst Man Cy B 42(3) (2012) 603-615

[21] Han LB, Xing KY, Chen X, Lei H, Wang F, Deadlock-free Genetic Scheduling for Flexible Manufacturing Systems using Petri Nets and Deadlock Controllers. Int J Prod Res 52(5) (2014) 1557-1572

[22] Luo JC, Xing KY, Zhou MC, Li XL, Wang XN, Deadlock-Free Scheduling of Automated Manufacturing Systems Using Petri Nets and Hybrid Heuristic Search. IEEE T Syst Man Cy-S 45(3) (2015) 530-541

[23] Baruwa OT, Piera MA, Guasch A, Deadlock-free Scheduling Method for Flexible Manufacturing Systems Based on the Timed Colored Petri nets and Anytime Heuristic Search. IEEE T Syst Man Cy-S 45(5) (2015) 831-846

[24] Lei H, Xing KY, Gao ZX, Xiong FL, A Hybrid Discrete Differential Evolution Algorithm for Deadlock-free Scheduling with Setup Times of Flexible Manufacturing Systems. T I Meas Control 38(10) (2016) 1270-1280

[25] Han, L. B., K. Y. Xing, X. Chen and F. L. Xiong, A Petri Net-based Particle Swarm Optimization Approach for Scheduling Deadlock-prone Flexible Manufacturing Systems. J Intell Manuf doi:10.1007/s 10845-015-1161-2

[26] Mühlenbein H, Paaß G, From Recombination of Genes to the Estimation of Distributions I. Binary Parameters. in: Parallel Problem Solving from Nature Springer, New York, (1998) 178-187

[27] Larranaga P, Lozano JA, Estimation of Distribution Algorithms: a new tool for evolutionary computation. Kluwer Academic Publishers, Boston, (2002)

[28] Jarboui B, Eddaly M, Siarry P, An Estimation of Distribution Algorithm for Minimizing the Total Flowtime in Permutation Flowshop Scheduling Problems. Comput Oper Res 36(9) (2009) 2638-2646

[29] Zhang Y, Li XP, Estimation of Distribution Algorithm for Permutation Flowshops with Total Flowtime Minimization. Comput Ind Eng 60(4) (2011) 706-718

[30] Ceberio J, Irurozki E, Mendiburu A, Lozano JA, A Distance-based Ranking Model Estimation of Distribution Algorithm for the Flowshop Scheduling Problem. IEEE T Evolut Comput 18(2) (2014) 286-300

[31] Liang XL, Chen HP, Lozano JA, A Boltzmann-based Estimation of Distribution Algorithm for a General Resource Scheduling Model. IEEE T Evolut Comput 19(6) (2015) 793-806

[32] Lozano JA, Larranaga P, Inza I, Bengoetxea E, Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms. Springer, New York, (2006)

[33] Murata T, Petri Nets: Properties, Analysis and Applications. In Proc IEEE 77 (1989) 541-580

[34] Zurawski R, Zhou MC, Petri nets and industrial applications: A tutorial. IEEE T Ind Electron, 41(6) (1994) 567-583

[35] Popova-Zeugmann L. Time and Petri Nets. Springer, Berlin Heidelberg, (2013)

[36] Hauschild M, Pelikan M, An Introduction and Survey of Estimation of Distribution Algorithms. Swarm Evolut Comput 1(3) (2011) 111-128

[37] Chang, PC, Chen SH, Liu CH, Sub-population Genetic Algorithm with Mining Gene Structures for Multiobjective Flowshop Scheduling Problems. Expert Syst Appl 33(3) (2007) 762-771

[38] Cormen TH, Leiserson CE, Rivest RL, Stein C, Introduction to Algorithms, 3rd edition. MIT Press, Massachusetts, (2009)

[39] Hollander M, Wolfe DA, Nonparametric Statistical Methods. Wiley, New York, (1973)