APEM journal
Advances in Production Engineering & Management
Volume 14 | Number 3 | September 2019 | pp 271–283
https://doi.org/10.14743/apem2019.3.327
ISSN 1854-6250
Journal home: apem-journal.org
Original scientific paper

Dynamic scheduling in the engineer-to-order (ETO) assembly process by the combined immune algorithm and simulated annealing method

Jiang, C.[a,*], **Xi, J.T.**[a]

[a]School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai, P.R. China

A B S T R A C T

With the increasing demand for customization, the engineer-to-order (ETO) production strategy plays an increasingly important role in today's manufacturing industry. The dynamic scheduling problem in ETO assembly process was investigated. We developed the mathematical model to represent the problem. In order to reduce rescheduling frequency, we introduced the concept of starting time deviation and improved the rolling horizon driven strategy. We proposed the hybrid algorithm combining immune algorithm (IA) and simulated annealing (SA) with the minimization of the rescheduling cost as the objective. The IA was designed as the global search process and the SA was introduced to improve the local searching ability. The scenario-based approach was used to model the disruptions affecting the tasks to be executed. Performance of the rolling horizon driven strategy and the hybrid algorithm were evaluated through simulations, the experiment analysis showed the best parameters of rolling horizon methods and demonstrated the feasibility of the hybrid algorithm. The hybrid algorithm was tested on different scale benchmark instances and the case that collected from a steam turbine assembly shop. The quality of solution in terms of cost obtained by the hybrid algorithm was found superior to the other three algorithms proposed in the literature.

A R T I C L E   I N F O

Keywords:
Engineer-to-order (ETO);
Assembly process;
Dynamic scheduling;
Rescheduling;
Rolling horizon;
Immune algorithm;
Simulated annealing

**Corresponding author:*
sjjzxjc@163.com
(Jiang, C.)

Article history:
Received 16 April 2019
Revised 14 September 2019
Accepted 16 September 2019

1. Introduction

The today's manufacturing industry is characterized by increasingly complex customized product, many companies have changed their business models from make-to-stock to X-to-order, where X usually stands for configure or engineer. Configure-to-order (CTO) is a modular approach, assembles orders from existing building blocks that can be delivered from stock and is hardly any engineering involved. The engineer-to-order (ETO) strategy is used when complex structures are needed to be built. The finished product and many components have never been built before and are impossible to be handled with standard variations. The ETO production strategy has become a trend [1,2]. During the ETO process, the most important phase is assembly process [3]. Assembly process accounts for almost 50 % of the total production time, 20 % of the total production cost and 30 % to 50 % of the labour cost [4]. The customer of ETO products are very strict with delivery deadline and late delivery will be punished. In ETO companies, several products are assembled in parallel [5] and the layout of typical ETO product assembly shop is shown in Fig. 1. Some workers formed the assembly group to execute the assembly task and the task duration depends on the number and skill level of workers [6]. The typical working situation of ETO assembly is shown in Fig. 2.

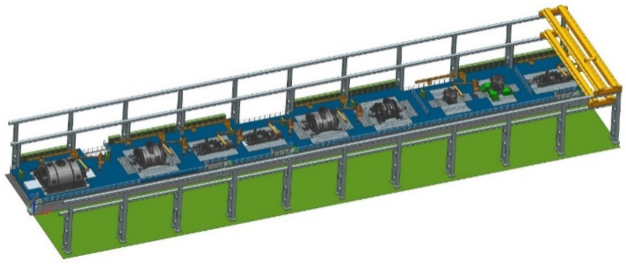


Fig. 1 The layout of typical ETO product assembly shop

Fig. 2 Typical working situation of ETO assembly

There are many unexpected events occur frequently during the ETO assembly process, such as the late delivery of necessary components, rework due to quality problems, etc [7,8]. There events usually exceed the task duration and cause the deviations from the initial schedule. Due to the characteristic of manual assembly and inherent flexibilities in the schedule, the small deviations can be absorbed by the initial schedule (The concept of 'small' depends on the question). If the deviation exceeds the threshold, the initial schedule became infeasible and rescheduling is needed. Since such unexpected events usually occur in the actual production, it is of practical significance to study the rescheduling problem in ETO assembly process.

The terms 'rescheduling' and 'dynamic scheduling' are often used interchangeably in the literature [9]. Vieira *et al*. [10] have classified the dynamic scheduling strategies into three types: (1) reactive scheduling which is to repair the schedule [11]; (2) proactive scheduling which is to create a schedule robust with unexpected events [12]; (3) study how rescheduling strategy affect the performance of manufacturing systems. The dynamic scheduling problem in ETO assembly process is treated as reactive scheduling in multi-mode resource-constrained multi-project scheduling problem (MRCMPSP). For the review of reactive scheduling in project scheduling (PSP), we refer to Herroelen and Leus [12,13]. In the field of reactive scheduling in single-mode resource-constrained single-project scheduling problem(RCPSP), Van de Vonder [14] and Van de Vonder [15] *et al*. have dealt with the problem of task duration variability. The literature on reactive scheduling in the multi-mode resource-constrained single-project scheduling problem(MRCPSP) is scare, Zhu proposed a branch-and-cut and constrained programming procedure for a general class of reactive problems [16], Deblaere proposed and evaluated a number of dedicated exact methods and tabu search to solve the reactive scheduling problem [11], Chakraborty formulated the discrete time based models and proposed the reactive scheduling for a single or a set of disruptions [17]. To the best of our knowledge, the literature on reactive scheduling in the multi-mode resource-constrained multi-project scheduling problem (MRCMPSP) is none.

The rescheduling problem is different from the initial planning problem because the rescheduling decision needed to be made in a timely manner. As the exact methods can only be used to solve small projects which have less than 20 tasks [18], the metaheuristics are more suitable [7,19]. Considering the convergence ability of the IA and the exploitation ability of SA, we proposed a hybrid algorithm which combined IA and SA to solve this problem. The rolling horizon rescheduling strategy is proposed to improve the computational speed [20]. The rest of this paper will be organized as the follows: Section 2 provides the mathematical model for the problem and section 3 describes the rolling horizon rescheduling strategy and the hybrid algorithm. In section 4, the hybrid algorithm and three other metaheuristic algorithms proposed in the literature are applied to solve benchmarks selected from literature and the industrial case. Conclusions and future directions are given in section 5.

2. Problem definition

The decision problem concerns with rescheduling tasks and their corresponding worker allocation in the multi-project environment. The objective is to minimize the rescheduling cost, which is the sum of the task starting time deviation cost, mode switching cost and tardiness cost. The deviation of task starting time will incur the cost of additional storage and crane for the required components. The mode switching cost is often regarded as "administrative" cost [11]. The tardiness cost is the penalties associated with late project completion [21]. In this study, we consider the problem subject to the following assumptions:

(1) Manual assembly processes are assumed to be carried out by a worker team. A mode represents a task-worker team with a constant duration.
(2) During the execution of each task, the assigned mode cannot be changed, i.e. preemption is not allowed during the execution of each task.
(3) The precedence relationships of each project force each task to be scheduled after all precedence tasks, the projects are independent of each other.
(4) Each worker cannot be allocated to more than one task at the same time.
(5) The maximum number of workers executing task is constrained by the work-space.
(6) The set-up time for each task is included in the task duration, and the transportation time of workers between the tasks is negligible.
(7) The rescheduled task starting time cannot be earlier than the task starting times from the initial schedule.

The notation used in this section can be summarized as follows:

Indices:

I	Set of projects
J_i	Set of tasks for project $i \in I$
Q	Set of maximum number of tasks for each project that can be executed concurrently due to floor space constraint
M_{ij}	Set of task execution modes in task $j \in J_i$, which correspond to the worker team
K	Set of hierarchical levels
T	Set of time periods
W	Set of workers

Parameters

r_i	release date of project $i \in I$, i.e. the earliest time that project $i \in I$ can start
d_i	due date of project $i \in I$
q_i	the maximum number of tasks in project i that can be executed concurrently due to floor space constraint
$pred(j)$	the predecessor set of task $j \in J_i$, i.e. $pred(j) = \{j' \vert j' \prec j\}$
w_{ijmax}	maximum number of workers executing task $j \in J_i$
w_{ijmin}	minimum number of workers executing task $j \in J_i$
w_k	number of type-k workers
z_{mk}	number of type-k workers in mode $m \in M_{ij}$
d_{ijm}	duration of task $j \in J_i$ in mode $m \in M_{ij}$
n_{ij}	unit cost of starting time deviation of task $j \in J_i$
$c_{ijm^*_{ij}}$	cost incurred by switching the mode from m_{ij} to m^*_{ij}
p_i	unit cost of violating the due date of project i
s_{ij}	start time of task $j \in J_i$ from the initial schedule
dur^*_{ij}	processing time of rescheduled task $j \in J_i$
f^*_{ij}	end time of rescheduled task $j \in J_i$

Decision variables:

s^*_{ij}	start time of task $j \in J_i$ after rescheduling

$$x^*_{ijt} = \begin{cases} 1, \text{if rescheduled task } j \text{ is performed at time } t \in (0, T] \\ 0, \text{otherwise} \end{cases}$$

$$y^*_{ijm} = \begin{cases} 1, \text{if rescheduled task } j \text{ is executed in mode } m \in M_{ij} \\ 0, \text{otherwise} \end{cases}$$

Under the assumptions and notations, the mathematical model is defined as follows:

Objective function:

$$\min\ C = \sum_{i=1}^{I}\sum_{j=1}^{J} n_{ij} \cdot (s_{ij}^{*} - s_{ij}) + \sum_{i=1}^{I}\sum_{j=1}^{J} c_{ijm_{ij}^{*}} + \sum_{i=1}^{I} p_i \cdot max(0, f_i^{*} - d_i) \tag{1}$$

Constraint conditions:

$$s_{ij}^{*} \geq r_i \quad \forall i \in I, j \in J_i \tag{2}$$

$$w_{ijmin} \leq \sum_{k=1}^{K} y_{ijm}^{*} \cdot z_{mk} \leq w_{ijmax} \quad \forall i \in I, j \in J_i, m \in M_{ij} \tag{3}$$

$$\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{m=1}^{M_{ij}} x_{ijt}^{*} \cdot y_{ijm}^{*} \cdot z_{mk} \leq w_k \quad \forall t \in (0, T], \forall k \in K \tag{4}$$

$$s_{ij'}^{*} + \sum_{m=1}^{M_{ij}} y_{ij'm}^{*} \cdot d_{ij'm} \leq s_{ij}^{*} \quad \forall i \in I, j \in J_i, j' \in pred(j) \tag{5}$$

$$\sum_{j=1}^{J} x_{ijt}^{*} \leq q_i \quad \forall i \in I, \forall t \in (0, T] \tag{6}$$

$$\sum_{m=1}^{M_{ij}} y_{ijm}^{*} = 1 \quad \forall i \in I, j \in J_i \tag{7}$$

$$dur_{ij}^{*} = \sum_{m=1}^{M_{ij}} y_{ijm}^{*} \cdot d_{ijm} \quad \forall i \in I, j \in J_i, m \in M_{ij} \tag{8}$$

$$f_{ij}^{*} = s_{ij}^{*} + dur_{ij}^{*} \quad \forall i \in I, j \in J_i \tag{9}$$

$$S_{ij}^{*} \geq S_{ij} \quad \forall i \in I, j \in J_i \tag{10}$$

The objective function in Eq. 1 is to minimize the total rescheduling cost C. Eq. 2 ensures that the rescheduled start time of each task for each project should not be earlier than the release date of project. Eq. 3 implies the number of workers assigned to each task must be within certain limits due to the work space constraint. Eq. 4 implies that at any time, the number of type-k workers assigned to the tasks should be lower than or equal to the total number of type-k workers. Eq. 5 ensures that the precedence relationships between tasks of the same project are not violated, the starting time of task should not be earlier than the finishing time of its predecessor set of tasks. Eq. 6 implies that number of tasks that can be executed concurrently of each project should be limited due to floor space constraint. Eq. 7 ensures that each task is being performed only in one mode (i.e. cannot change or interrupt the execution mode). Eq. 8 denotes that the processing time of task equals to the processing time of assigned execution mode for that task. Eq. 9 implies that the end time of task equals to the start time plus the processing time, that means, the task is non-preemptive. Eq. 10 ensures that the rescheduled task starting time should be at least greater or equal to the task starting times from the initial schedule.

3. Materials and methods

3.1 Rolling horizon rescheduling strategy

The rolling horizon strategy is very suitable for solving large-scale scheduling problem, because it can optimize the system in a limited time horizon instead of globally optimizing the system in order to reduce the computational complexity. It can divide the original scheduling horizon into some periods, at each decision point, scheduling task set should be selected and local schedule should be made. As time horizon rolls forward sequentially, each local scheduling problem would be solved until the global schedule is gotten. The rescheduling strategy is shown in Fig. 3. The task starting time deviation is detected and judged by the rescheduling mechanism for whether to reschedule or not. If rescheduling is needed, the window is selected based on the window mechanism. The hybrid algorithm is performed within the window. The rescheduling mechanism and window mechanism are the two key elements of rolling horizon rescheduling strategy.

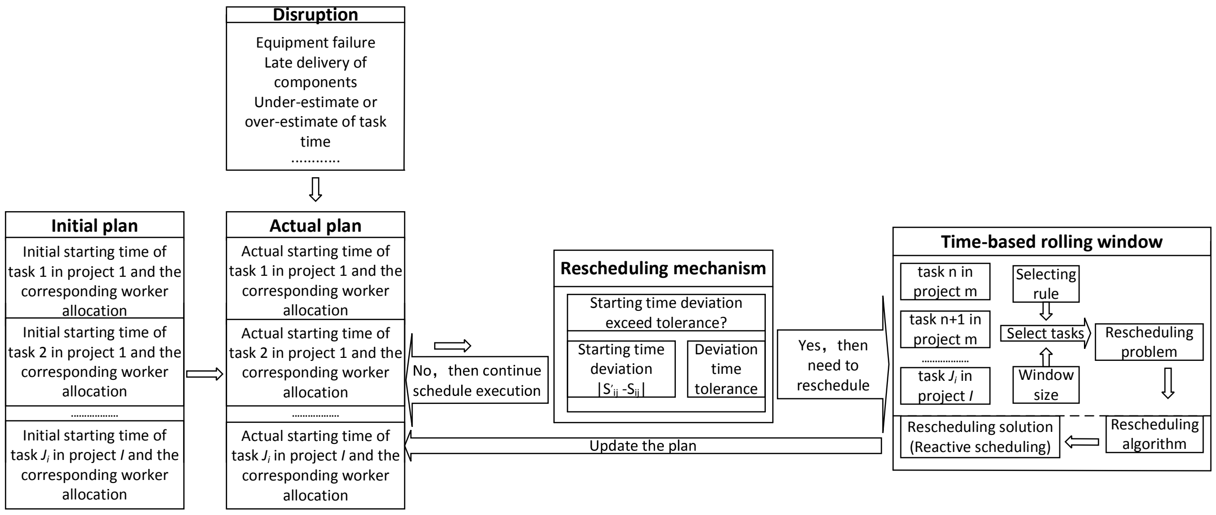

Disruption
Equipment failure
Late delivery of components
Under-estimate or over-estimate of task time
....
Initial plan
Initial starting time of task 1 in project 1 and the corresponding worker allocation
Initial starting time of task 2 in project 1 and the corresponding worker allocation
Initial starting time of task J_l in project l and the corresponding worker allocation
Actual plan
Actual starting time of task 1 in project 1 and the corresponding worker allocation
Actual starting time of task 2 in project 1 and the corresponding worker allocation
Actual starting time of task J_l in project l and the corresponding worker allocation
No, then continue schedule execution
Rescheduling mechanism
Starting time deviation exceed tolerance?
Starting time deviation $|S'_{ij} - S_{ij}|$
Deviation time tolerance
Yes, then need to reschedule
Time-based rolling window
task n in project m
task n+1 in project m
task J_l in project l
Selecting rule
Select tasks
Window size
Rescheduling problem
Rescheduling algorithm
Rescheduling solution (Reactive scheduling)
Update the plan

Fig. 3 Rolling rescheduling strategy

In the actual production, the deviation of task processing time is inevitable. This will lead to deviation between actual starting time and initial starting time of successor task, then the initial schedule may be infeasible and the rescheduling is required [21]. If each deviation is adjusted, it will lead to frequent rescheduling and reduce the efficiency of production. We established the buffer mechanism and introduced the concept of starting time deviation tolerance, it can filtrate the small deviation and consider the accumulation caused by a large number of small deviations. The concept of task starting time deviation is proposed, and given by:

$$dev_i = s'_{ij} - s_{ij}, j = arg_{(j)}\max\{s_{ij} | s'_{ij} \le T, \forall i, \forall j\} \tag{11}$$

$$\delta = deviation = max(dev_i) \ , \forall i \tag{12}$$

Each task in multi projects may have deviation, then we select the current maximum deviation δ. We determine the task starting time deviation tolerance δ_{max}. During the process, δ is compared with δ_{max}. Once δ exceeds δ_{max}, the rescheduling is needed. The number of δ_{max} is very important, if it is too large, it can't respond to the disruptions quickly; if it is too small, it will incur frequent rescheduling.

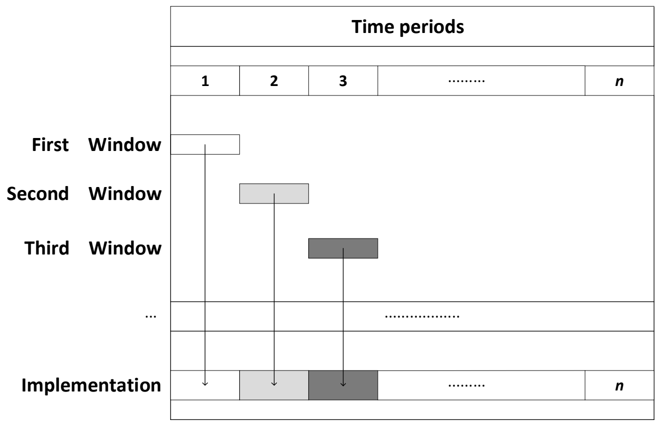


Fig. 4 Time-based rolling window

We select the rescheduling window mechanism based on time. We divide the original scheduling horizon $[0, K]$ into n periods with equal length, as in Fig. 4 shows. The window size has a great impact on computational efficiency. If the window is too large, it is difficult to obtain the optimal solution as the computational complexity is large; If it is too small, the considered global information is less and the global optimal performance may not be satisfied.

3.2 Solution procedure

The immune algorithm (IA) is a population-based optimization algorithm introduced by De Castro [22]. It has the merits of easy implementation, fast convergence. However, IA is easily trapped into local optima. Simulated annealing (SA) algorithm is a local search metaheuristic proposed by Metropolis, Nicholas [23]. The convergence properties and hill-climbing moves to escape local optima have made it become the popular local search algorithm [24]. In this study, we proposed a hybrid algorithm combining the merits of IA and SA algorithm for solving this problem. The proposed algorithm consists of two phases, the first phase is IA which is used for the global searching process and the solution is treated as the initial solution in the second phase. The second phase is SA, which is employed for the local searching process. The framework of the proposed algorithm is shown in Fig. 5.

Initialization

Step 1: Combining all the projects and relabel the tasks

We combined all the projects into a combined precedence graph and re-numbered the tasks. If the first project has n tasks, then the first task of second project is re-numbered as $n + 1$, and so on.

Step 2: Encoding and decoding

We adopted the chromosome encoding method. The length of chromosome is twice the number of total tasks. The representation is comprised of two vectors, the first is the precedence feasible task list (TL) for scheduling process, while the second vector is the mode assignment for tasks execution. The TL is a precedence feasible permutation of tasks, in which each task must occur after all its predecessors and before all its successors. The second vector is a list of execution modes for all tasks, the k-th element of this list defines the execution mode of task k. Each chromosomal representation determines the sequence of tasks for each project and the mode for each task. We applied the parallel schedule generation scheme (parallel SGS) to generate the schedule related to individuals, which consist of precedence feasible TL with the mode assignment.

Immune algorithm phase

Immune algorithm (IA) is inspired by the biological immune system defending the body from infection and disease. The immune system firstly recognizes toxins or bacteria as antigens, then generate a set of antibodies to eliminate the antigens. The antibodies which are better at eliminating the antigens will have more variants in the next generation. Each antibody is assigned a value called affinity showing the ability to eliminate antigens. The antigen, affinity and antibody in the IA are equivalent to the problem to be solved, objective function and feasible solution.

Step 1: Identify antigen and generate initial population

The optimization problem needs to be transformed to the form that the algorithms can identify and evolve. Each antibody is designed to represent a feasible solution of the problem and the corresponding problem is the antigen. Then, generate the initial population randomly without any experience.

Step 2: Evaluate the affinity value of antibodies

The affinity value is used as the performance evaluation of each antibody. The antibodies with higher affinity value are better at eliminating antigens. Eq.13 is used to define the affinity value.

$$Affinity = \frac{1}{Cost} \tag{13}$$

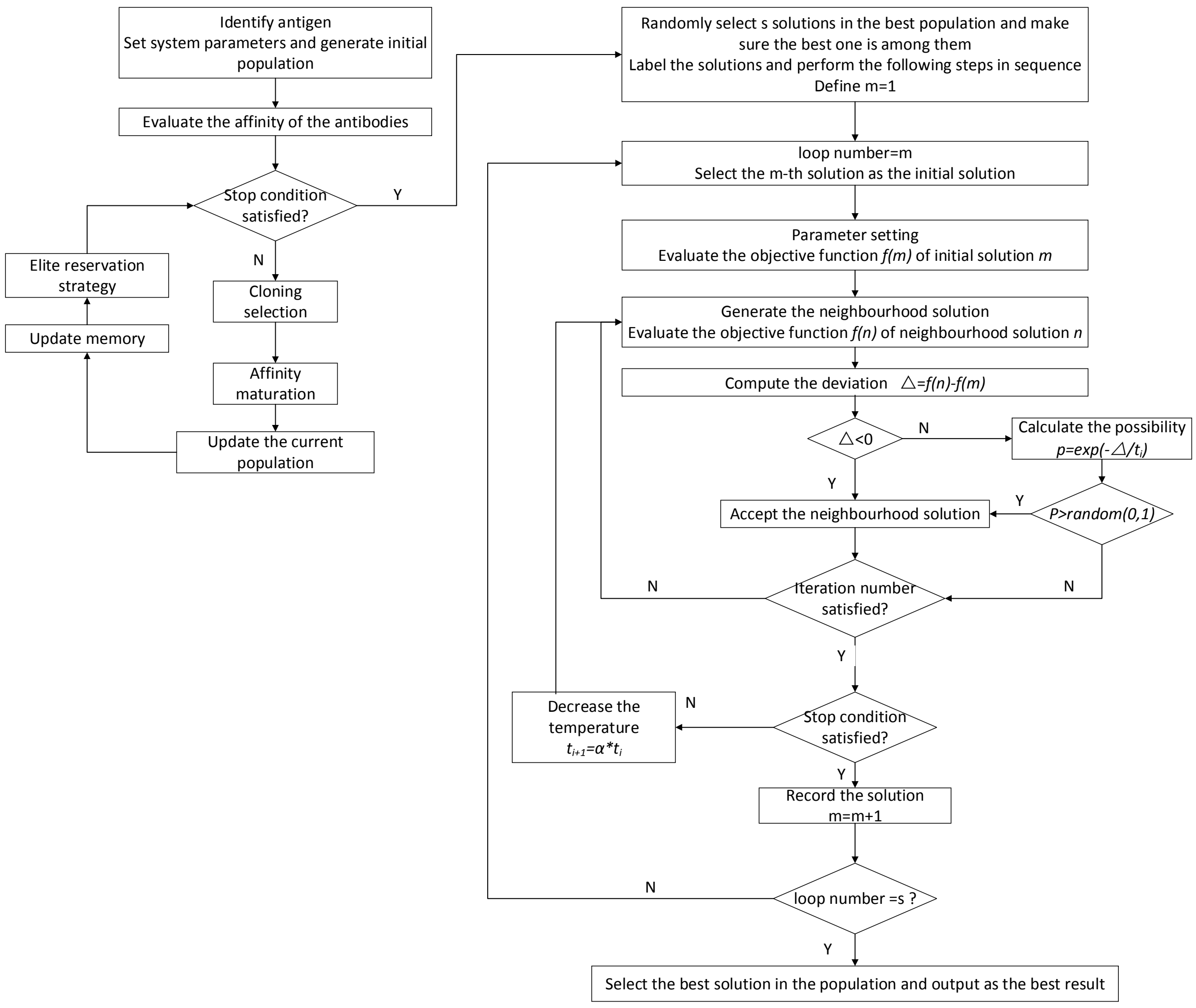

Identify antigen
Set system parameters and generate initial population
Evaluate the affinity of the antibodies
Stop condition satisfied?
Y
N
Elite reservation strategy
Cloning selection
Update memory
Affinity maturation
Update the current population
Randomly select s solutions in the best population and make sure the best one is among them
Label the solutions and perform the following steps in sequence
Define m=1
loop number=m
Select the m-th solution as the initial solution
Parameter setting
Evaluate the objective function f(m) of initial solution m
Generate the neighbourhood solution
Evaluate the objective function f(n) of neighbourhood solution n
Compute the deviation △=f(n)-f(m)
△<0
Calculate the possibility p=exp(-△/t_i)
Accept the neighbourhood solution
P>random(0,1)
Iteration number satisfied?
Decrease the temperature t_{i+1}=α*t_i
Stop condition satisfied?
Record the solution m=m+1
loop number =s ?
Select the best solution in the population and output as the best result

Fig. 5 Framework of the hybrid algorithm

Step 3: Cloning selection

The cloning selection is proportional to the affinity value and the better antibodies will have more chance of being proliferated. All the proliferated antibodies are assembled in a mutating pool and the antibodies with higher affinity would have more clones. The cloning selection probability of each antibody is calculated as shown in Eq. 14:

$$Rate\ of\ cloning(ROC) = \frac{Affinity\ value\ of\ solution}{Total\ affinity\ value\ of\ solution\ in\ the\ population} \tag{14}$$

Step 4: Affinity maturation

The affinity maturation is to make random changes in the proliferated antibodies in order to generate better antibodies. The new solutions are structurally and behaviourally similar to their creators but not the exactly same. The clones with different affinity suffer from different rate of change. The clones with higher affinity will suffer a slight change and the clones with lower affinity will suffer a higher change.

If the termination criterion satisfies, usually a special number of generations or a sufficiently good fitness, then stop IA phase, otherwise continue. The IA phase could provide diverse and elite initial solutions for the SA phase.

Simulated annealing phase

Simulated annealing (SA) is a method which models the physical process of heating a material and then slowly lowering the temperature until the lowest-energy state is reached. At each virtual annealing temperature, the simulated annealing algorithm generates a neighbour solution to the problem. The acceptance of the neighbour solution is based on the satisfaction of the Metropolis criterion, and this procedure is iterated until convergence.

Step 1: Initialization

Set the parameters, the number of stages (S) and number of iterations (I). In our algorithm, the initialization solution of SA phase is provided by IA phase. Individuals of the best population were randomly selected, making sure that the global best solution was included. The solutions were numbered and the following steps were performed in sequence.

Step 2: The neighbourhood solution generation

The neighbourhood structure plays a very important role in a local search. It is a mechanism which can apply a small perturbation to the given solution in order to obtain a new set of neighbouring solutions. We implemented three different neighbourhood operators proposed in our previous work [25].

Step 3: Evaluation and Comparison

Evaluate the objective function of initial solution and neighbour solution. The deviation between objective function of two candidate solutions is computed as $\Delta = f(n) - f(s)$. If $\Delta \leq 0$, the neighbour solution n is accepted. Otherwise, generate a random number $r \in [0,1]$ and accepting the neighbour solution n if $r < \exp(-\Delta/t_i)$. t_i is the current temperature.

Step 4: Termination criterion

At each iteration, the temperature is fixed. If the number of completed iterations is equal to I and terminate the current stage. The temperature decreases from one stage to another under the cooling schedule mechanism. We adopt the exponential cooling schedule, the temperature of i-th stage is calculated as $t_i = \alpha * t_{i-1}$ and $\alpha \in (0,1)$ is the temperature decreasing rate.

The termination criterion was used to determine whether the proposed method should stop. If the annealing temperature is reduced to the extreme point the stop criterion is fulfilled.

4. Results and discussion

In this section, we presented the results of computational studies and evaluate the performance of the proposed algorithm. The algorithms were coded in MATLAB R2016a, running on the personal computer configured with 16GB memory and Intel (R)Core (TM)i7-6700. Instances in this paper come from the MISTA 2013 Challenge, which combine multiple MRCPSP instances from Project Scheduling Problem Library (PSPLIB) [26]. We introduce randomly generated disruption scenarios with the method proposed in the literature [11].

4.1 Parameters analysis

We set up experiments to analyse the impact of starting time deviation tolerance and horizon window size. We select the instance A5 and the makespan of initial schedule is 258. Randomly select 10 tasks in the initial schedule, and a duration increase Δ_{ij} is generated as a uniformly distributed random variable from the interval $[1, 0.3 * d_{ij}]$, as the maximal magnitude of the disruption equal 30 % of the deterministic task duration d_{ij}. Ten times calculation under different deviation tolerance are taken to solve the above example. The cost, rescheduling times and the average computation time are shown in Table 1. It can be seen that when the starting time deviation tolerance is 3, the cost is minimized. Ten times calculation under different size of horizon window are taken to solve the above example, in which the starting time deviation tolerance is 3. The cost, rescheduling times and the average computation time are shown in Table 2. When the horizon window size is 20, the solution is optimal.

Table 1 The results of the starting time deviation tolerance experiments

The tolerance of starting time deviation	Cost	Rescheduling times	Computation time(s)
1	225.1	12	132.9
2	128.3	11	115
3	94.9	10	100.7
4	118.1	9	92.4
5	178.5	8	76.3
6	212.7	6	66.2
7	258.7	5	56.6
8	335.4	4	38.6
9	369.2	2	24.8
10	458.5	1	12.6
11	540	0	0
12	540	0	0

Table 2 The results of the size of horizon window experiments

The size of horizon window	Cost	Rescheduling times	Computation time(s)
10	293.4	19	78.2
15	102.5	13	92.2
20	68.1	10	99.8
25	107.9	8	109.5
30	202.2	7	124.1
60	684.7	4	187.7
90	896.9	3	277.1
120	1397.7	2	302
150	1870.2	2	372.9
180	2137.5	2	494
210	2336.8	1	529.6
230	2583.4	1	567.2

4.2 Performance evaluation

Ten instances are selected from MISTA 2013 Challenge and introduced the disruption scenarios to model the uncertainty events. The details of ten instances is shown in Table 3. To evaluate the proposed algorithm, comparisons were made with Immune Algorithm (IA) proposed by Mobini [27], Simulated Annealing (SA) proposed by Józefowska [28] and Genetic Algorithm (GA) proposed by Goncharvo [29]. The parameters of GA, IA, and SA are set by trial and error. The scheduling results of IA-SA, GA, IA, and SA are shown in Table 4, including the best makespan (BST) and the average makespan (AVG) on ten independent runs. The best BST results of each instance are set in bold. From Table 4, it is shown that the BST and AVG values obtained by IA-SA are better than those obtained by other three algorithms on all instances, which demonstrate IA-SA has the superiority of searching quality and robustness. The average computation time of four algorithms is shown in Table 5 and illustrates how the computational time of different methods changes with the increase of problem sizes.

Table 3 Benchmark instances for dynamic scheduling algorithm

Instance	Number of projects	Total number of tasks	Number of disruption scenarios	The number of concurrent execution tasks in each project
A3	2	60	6	2
A6	5	150	15	3
A7	10	100	10	2
B2	10	200	20	3
B4	15	150	15	2
B5	15	300	30	3
B6	15	450	45	2
B10	20	420	42	3
X8	20	400	40	2
X9	20	600	60	3

Table 4 Cost of four algorithms over ten runs

Instance	IA		SA		GA		IA-SA	
	AVG	BST	AVG	BST	AVG	BST	AVG	BST
A3	52.5	42	66.4	51	62.9	49	42.4	**37**
A6	228	200	292.9	203	282.9	184	173.9	**161**
A7	236.3	219	288.7	217	271	211	170.3	**139**
B2	335	308	395.6	365	370.7	324	312.2	**301**
B4	426.5	325	475.1	433	437.7	354	396.4	**298**
B5	584.7	519	721.3	679	690.6	616	541.7	**510**
B6	1043	910	1278	989	1035	936	972	**894**
B10	1398	1293	1524	1387	1376	1295	1301	**1277**
X8	1045	982	1321	1019	992	926	958	**905**
X9	1802	1703	2317	2013	1779	1695	1722	**1658**

Table 5 Average computation time of four algorithms

Instance	Computation time(s)			
	IA	SA	GA	IA-SA
A3	24	18.3	37.5	29.4
A6	42.5	30.4	62.7	48.4
A7	38.5	27.1	52.3	40.5
B2	59.2	39.2	81.9	63
B4	36.2	26.2	50.1	39.8
B5	98.6	78.6	165.3	102.6
B6	192.5	142.2	291.7	208.2
B10	178.5	139.6	269.3	185.1
X8	207.8	152.3	305.2	218.4
X9	278.1	203.5	425.2	301.9

Furthermore, to compare results obtained in different instances, relative percentage deviation (RPD) is introduced as the only dependent variable of variance analysis, as shown in Eq.15, Where Alg_{sol} represents the objective value obtained by single algorithm running, and BST_{sol} represents the best solution over the whole set of results concerning the same instance. Obviously, the smaller RPD value is, the better the result is.

$$RPD = \frac{Alg_{sol} - BST_{sol}}{BST_{sol}} \times 100 \tag{15}$$

The ANOVA and Least-Significant Difference (LSD) tests were conducted in SPSS to check the results transformed into RPD value. Test results revealed that under confidence interval of 95 %, the p value was 0, which means there are significant differences in performance of the four algorithms. Fig. 6 depicts mean plot with LSD intervals for RPD value obtained by four algorithms. In this measure, the proposed algorithm outperforms the other three algorithms.

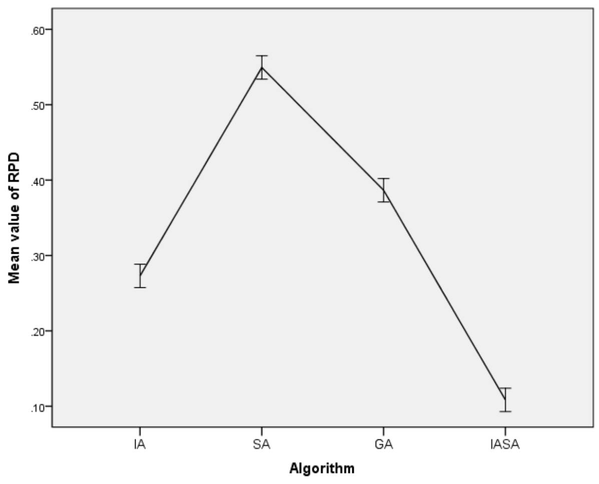


Fig. 6 Mean plot and LSD intervals for algorithms in RPD value

4.3 Case study

To demonstrate the applicability of the proposed method from a practical point of view, a case that uses industrial data from a collaborating steam turbine company was considered. The case was also shown in our previous work [25]. We run PSO-TS algorithm in our previous work ten times and select the best schedule as the initial schedule. The makespan of the initial schedule is 1524 hours. To model the uncertainty, a set of disruption scenarios has been introduced and extended the task duration based on historical production records. Some task processing time variations are higher than the deviation tolerance, while some are lower than the deviation tolerance. The processing time variations have been accumulated. The scenarios defined only for the assembling and testing process which often exceed the time limit. Then we run four rescheduling algorithms ten independent runs and select the best result. The makespan of the IA algorithm is 1913 hours (cost: 4920). The makespan of the SA algorithm is 1938 hours (cost: 7012). The makespan of the GA algorithm is 1912 hours (cost: 5033). The Gantt chart of the IA-SA algorithm can be seen in Fig. 7, in which the makespan is 1887 hours (cost: 4308). In Fig. 7, the horizontal axis represents the time horizon, the vertical axis represents the project number, the box represents the task, the number in the box represents the task number and the length represents the duration of the task. We could see the sequence, the duration of tasks in each project and some tasks can be executed concurrently under space constraints. The difference between rescheduled makespan and initial makespan is less than the accumulated processing time variations. Comparison of the four algorithms can be seen in Fig. 8(a), 8(b). It is clear that IA-SA algorithm outperforms the other three algorithms and can be assisted in the dynamic scheduling of steam turbines assembly process.

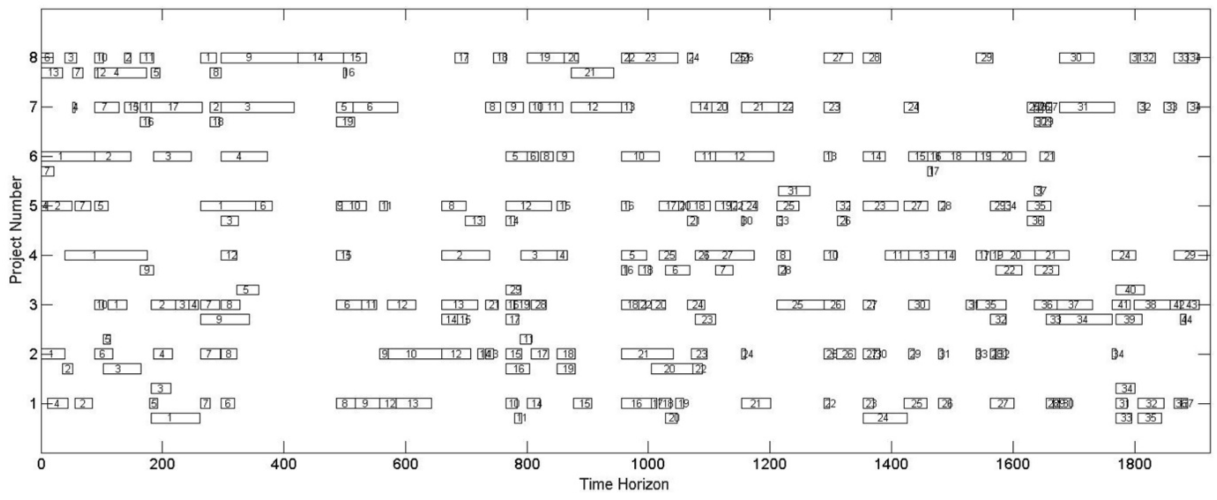
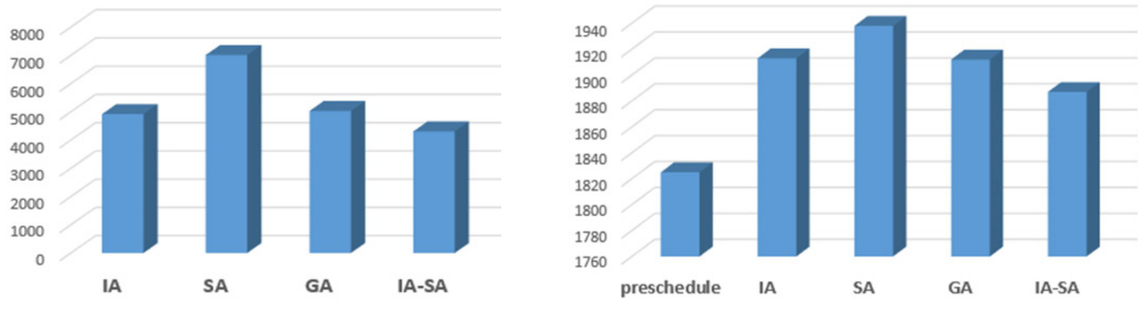


Fig. 7 Gantt chart of IA-SA algorithm



(a) (b)

Fig. 8 Comparison of makespan (a), and comparison of cost (b)

5. Conclusion

In order to overcome the dynamic scheduling problem in ETO assembly process, we proposed the hybrid algorithm based on rolling horizon strategy. After introduced the concept of task starting time deviation and designed the rolling horizon strategy based on the tolerance of starting time deviation, we improved the traditional rolling horizon strategy to avoid frequent rescheduling. The time-based rescheduling window mechanism was designed in order to realize the buffer of dynamic event. To guarantee the computational efficiency of the rolling rescheduling algorithm, based on the rolling horizon procedure, the hybrid approach combining immune algorithm and simulated annealing algorithm was proposed, and tested on different scale benchmark instances and industrial case. The computational results demonstrated the superiority of proposed algorithm.

Directions for future research can be outlined as follows: Firstly, the computational time of the proposed hybrid approach grows with the size of instances, the more efficient algorithms should be considered. Secondly, the problem we studied has not taken the multi-objectives into account, such as total tardiness, the deviation from the original plan should be considered. Thirdly, some heavy and large components needed to be transported by crane, thus the dynamic scheduling and crane rescheduling should be dealt with together.

References

[1] Jana, T.K., Saha, P., Sarkar, B., Saha, J. (2013). Implementation of agent based holonic control in discrete manufacturing, *Advances in Production Engineering & Management*, Vol. 8, No. 3, 157-168, doi: 10.14743/apem2013.3.163.

[2] Grabenstetter, D.H., Usher, J.M. (2015). Sequencing jobs in an engineer-to-order engineering environment, *Production & Manufacturing Research*, Vol. 3, No. 1, 201-217, doi: 10.1080/21693277.2015.1035461.

[3] Alfieri, A., Tolio, T., Urgo, M. (2012). A project scheduling approach to production and material requirement planning in manufacturing-to-order environments, *Journal of Intelligent Manufacturing*, Vol. 23, No. 3, 575-585, doi: 10.1007/s10845-010-0396-1.

[4] De Lit, P., Latinne, P., Rekiek, B., Delchambre, A. (2001). Assembly planning with an ordering genetic algorithm, *International Journal of Production Research*, Vol. 39, No. 16, 3623-3640, doi: 10.1080/00207540110056135.

[5] Alfieri, A., Tolio, T., Urgo, M. (2011). A two-stage stochastic programming project scheduling approach to production planning, *The International Journal of Advanced Manufacturing Technology,* Vol. 62, No. 1-4, 279-290, doi: 10.1007/s00170-011-3794-4.

[6] Hytonen, J., Niemi, E., Toivonen, V. (2008). Optimal workforce allocation for assembly lines for highly customised low-volume products, *International Journal of Services Operations and Informatics*, Vol. 3, No. 1, 28-39, doi: 10.1504/ijsoi.2008.017703.

[7] Jiang, P., Ding, J.L., Guo, Y. (2018). Application and dynamic simulation of improved genetic algorithm in production workshop scheduling, *International Journal of Simulation Modelling*, Vol. 17, No. 1, 159-169, doi: 10.2507/IJSIMM17(1)CO3.

[8] Yang, X.P., Gao, X.L. (2018). Optimization of dynamic and multi-objective flexible job-shop scheduling based on parallel hybrid algorithm, *International Journal of Simulation Modelling,* Vol. 17, No. 4, 724-733, doi: 10.2507/IJSIMM17(4)CO19.

[9] Hicks, C., Song, D.P., Earl, C.F. (2007). Dynamic scheduling for complex engineer-to-order products, *International Journal of Production Research*, Vol. 45, No. 15, 3477-3503, doi: 10.1080/00207540600767772.

[10] Vieira, G.E., Herrmann, J.W., Lin, E. (2003). Rescheduling manufacturing systems: A framework of strategies, policies, and methods, *Journal of Scheduling*, Vol. 6, No. 1, 39-62, doi: 10.1023/A:1022235519958.

[11] Deblaere, F., Demeulemeester, E., Herroelen, W. (2011). Reactive scheduling in the multi-mode RCPSP, *Computers & Operations Research*, Vol. 38, No. 1, 63-74, doi: 10.1016/j.cor.2010.01.001.

[12] Herroelen, W., Leus, R. (2004). Robust and reactive project scheduling: A review and classification of procedures, *International Journal of Production Research*, Vol. 42, No. 8, 1599-1620, doi: 10.1080/00207540310001638055.

[13] Herroelen, W., Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials, *European Journal of Operational Research*, Vol. 165, No. 2, 289-306, doi: 10.1016/j.ejor.2004.04.002.

[14] Demeulemeester, E., Herroelen, W., Leus, R. (2008). Proactive-reactive project scheduling, In: Artigues, C., Demassey, S., Néron, E. (eds.), *Resource-constrained project scheduling: Models, algorithms, extensions and applications,* Wiley-ISTE, London, United Kingdom, 203-211, doi: 10.1002/9780470611227.ch13.

[15] Van de Vonder, S., Ballestín, F., Demeulemeester, E., Herroelen, W. (2007). Heuristic procedures for reactive project scheduling, *Computers & Industrial Engineering*, Vol. 52, No. 1, 11-28, doi: 10.1016/j.cie.2006.10.002.

[16] Zhu, G., Bard, J.F., Yu, G. (2005). Disruption management for resource-constrained project scheduling, *Journal of the Operational Research Society*, Vol. 56, No. 4, 365-381, doi: 10.1057/palgrave.jors.2601860.

[17] Chakrabortty, R.K., Sarker, R.A., Essam, D.L. (2016). Multi-mode resource constrained project scheduling under resource disruptions, *Computers & Chemical Engineering*, Vol. 88, 13-29, doi: 10.1016/j.compchemeng.2016.01.004.

[18] Sonmez, R., Uysal, F. (2015). Backward-forward hybrid genetic algorithm for resource-constrained multiproject scheduling problem, *Journal of Computing in Civil Engineering*, Vol. 29, No. 5, Article number: 04014072, doi: 10.1061/(ASCE)CP. 1943-5487.0000382.

[19] Gholamian, M.R., Heydari, M. (2017). An inventory model with METRIC approach in location-routing-inventory problem, *Advances in Production Engineering & Management*, Vol. 12, No. 2, 115-126, doi: 10.14743/apem 2017.2.244.

[20] Qin, W., Zhang, J., Song, D. (2018). An improved ant colony algorithm for dynamic hybrid flow shop scheduling with uncertain processing time, *Journal of Intelligent Manufacturing*, Vol. 29, No. 4, 891-904, doi: 10.1007/s10845-015-1144-3.

[21] Deblaere, F., Demeulemeester, E., Herroelen, W. (2011). Proactive policies for the stochastic resource-constrained project scheduling problem, *European Journal of Operational Research*, Vol. 214, No. 2, 308-316, doi: 10.1016/j.ejor.2011.04.019.

[22] De Castro, L.N., Von Zuben, F.J, (1999). Artificial immune systems: Part I – Basic theory and applications, Technical Report, Technical report, RT DCA 01/99, 95 pages.

[23] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E. (1953). Equation of state calculations by fast computing machines, *The Journal of Chemical Physics*, Vol. 21, No. 6, 1087-1092, doi: 10.1063/1.1699114.

[24] Cao, Q.K., Qin, M.N., Ren, X.Y. (2018). Bi-level programming model and genetic simulated annealing algorithm for inland collection and distribution system optimization under uncertain demand, *Advances in Production Engineering & Management*, Vol. 13, No. 2, 147-157, doi: 10.14743/apem2018.2.280.

[25] Jiang, C., Hu, X., Xi, J. (2019). Integrated multi-project scheduling and hierarchical workforce allocation in the ETO assembly process, *Applied Sciences*, Vol. 9, No. 5, 885-904, doi: 10.3390/app9050885.

[26] Wauters, T., Kinable, J., Smet, P., Vancroonenburg, W., Vanden Berghe, G., Verstichel, J. (2016). The multi-mode resource-constrained multi-project scheduling problem, *Journal of Scheduling*, Vol. 19, No. 3, 271-283, doi: 10.1007/s10951-014-0402-0.

[27] Mobini, M., Mobini, Z., Rabbani, M. (2011). An artificial immune algorithm for the project scheduling problem under resource constraints, *Applied Soft Computing*, Vol. 11, No. 2, 1975-1982, doi: 10.1016/j.asoc.2010.06.013.

[28] Józefowska, J., Mika, M., Różycki, R., Waligóra, G., Węglarz, J. (2001). Simulated annealing for multi-mode resource-constrained project scheduling, *Annals of Operations Research*, Vol. 102, No. 1-4, 137-155, doi: 10.1023/A:1010954031930.

[29] Goncharov, E.N., Leonov, V.V. (2017). Genetic algorithm for the resource-constrained project scheduling problem, *Automation and Remote Control*, Vol. 78, No. 6, 1101-1114, doi: 10.1134/S0005117917060108.