Knowledge-Based Systems 72 (2014) 28–36

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Solving the steelmaking casting problem using an effective fruit fly optimisation algorithm

Jun-qing Li [a,b,*], Quan-ke Pan [d,*], Kun Mao [a], P.N. Suganthan [c]

[a] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[b] State Key Laboratory of Synthetic Automation for Process Industries, Northeastern University, ShenYang 110819, PR China
[c] Nanyang Technological University, Singapore 639798, Singapore
[d] State Key Lab of Digital Manufacturing Equipment & Technology in Huazhong University of Science & Technology, Wuhan, 430074, PR China

ARTICLE INFO

Article history:
Received 7 February 2014
Received in revised form 11 August 2014
Accepted 28 August 2014
Available online 6 September 2014

Keywords:
Hybrid flow shop scheduling
Steelmaking casting problem
Fruit fly optimisation algorithm
Realistic scheduling problem
Neighbourhood structure

ABSTRACT

This paper presents an effective fruit fly optimisation algorithm (FOA) to solve the steelmaking casting problem. First, we model the realistic problem as a hybrid flow shop (HFS) scheduling problem with batching in the last stage. Next, the proposed FOA algorithm is applied to solve the realistic HFS problems. In the proposed algorithm, each solution is represented by a fruit fly. Each fruit fly first improves its status through a well-designed smell search procedure. During the vision-based search procedure, the worst fruit fly in the population will be induced by the best fruit fly found thus far to improve the exploitation ability of the entire fruit fly population further. To enhance the exploration ability of the proposed algorithm, in each generation, each fruit fly that has not updated its status during the last several iterations will be replaced by a newly-generated fruit fly. The proposed algorithm is tested on sets of the instances that are generated based on the realistic production. Moreover, the influence of the parameter setting is also investigated using the Taguchi method of the design-of-experiment (DOE) to determine the suitable values for the key parameters. The results indicate that the proposed FOA is more effective than the four presented algorithms.

1. Introduction

In modern manufacturing and production systems, production scheduling plays an important role, which can obviously increase the production efficiency and profit. The hybrid flow shop (HFS) scheduling problem is one branch of the classical flow shop scheduling problem (FSSP), which has been verified to be an NP-hard problem [1]. The HFS scheduling problem is harder than the FSSP because of the addition of the consideration of parallel machine selection for each job. In 1988, Gupta proved that HFS is also an NP-hard problem [1]. Recent and comprehensive reviews on HFS can be found in [2,3]. The realistic HFS is a much more complex generalisation of the traditional HFS, which has been applied in the paper industry, carpet-manufacturing industry, and tile industry [4]. Voss and Witt considered a real-world multi-mode multi-project scheduling problem, in which the resources form a hybrid flow shop consisting of 16 production stages [5]. Ruiz et al. solved the realistic HFS problems with skipped stages, sequence-dependent setup times, machine lags, release dates, machine eligibility and precedence relationships [6]. Behnamian et al. presented a Pareto-based random key genetic algorithm for minimising the makespan and sum of the earliness and tardiness of jobs [7]. Dugardin et al. focused on the multi-objective resolution of a re-entrant HFS [8]. More recently, Costa et al. presented a dual encoding-based meta-heuristic algorithm for solving a constrained hybrid flow shop scheduling problem [9]. Figielska proposed a heuristic for scheduling in a two-stage HFS with renewable resources shared among the two stages [10]. Luo et al. studied the HFS in an active environment, where family setup time and inconsistent family formation were considered [11]. Wang et al. considered the HFS in a solar cell industry, where dedicated machines and lot-splitting were considered [12]. Chung and Liao solved the HFS by an immunoglobulin-based artificial immune algorithm [13]. Chou proposed a particle swarm optimisation (PSO) with cocktail decoding method for the HFS problems with multi-processor tasks [14]. Yang solved the HFS with two stages and dedicated machines at the first stage [15].

The steelmaking problem is one of the most complex realistic HFS, which is accompanied by technological constraints of steelmaking. In the iron and steel industries, the steelmaking process is usually the bottleneck and the crucial issue to improve the

* Corresponding authors at: State Key Lab of Digital Manufacturing Equipment & Technology in Huazhong University of Science & Technology, Wuhan, 430074, PR China. Tel.: +86 18606356701.

E-mail addresses: lijunqing.cn@gmail.com (J.-q. Li), panquanke@gmail.com (Q.-k. Pan).

http://dx.doi.org/10.1016/j.knosys.2014.08.022
0950-7051/

productivity of the production system [16]. Xuan and Tang modelled the steelmaking process as a complex HFS problem with batch production at the last stage [17]. Tang et al. presented a non-linear model based on the just-in-time (JIT) concept for solving machine conflicts in steelmaking-continuous casting [18]. Atighehchian et al. investigated a novel iterative algorithm combining ant colony optimisation (ACO) and non-linear optimisation methods [19]. Missbauer et al. divided the scheduling task into four sub-problems and presented a heuristic algorithm consisting of three planning levels [20]. More recently, Tan and Liu studied the problem to reduce the electricity cost and the associate production cost [21]. Tang et al. investigated the slab reallocation problem arising from operations planning in the steel industry by a heuristic algorithm based on tabu search (TS) [22]. Xuan and Li investigated the batch decomposition strategy by a mixed backward and forward dynamic programming approach [23]. Pan et al. proposed an efficient artificial bee colony (ABC) algorithm for solving the steelmaking problem by assigning a different penalty coefficient for three objectives, i.e., the average sojourn time, and the earliness/tardiness penalty [4]. Mao et al. investigated the HFS in the steelmaking-continuous casting process using a Lagrangian relaxation approach [24].

Very recently, by mimicking the food search procedure from the fruit fly swarm, a novel swarm optimisation algorithm named fruit fly optimisation algorithm (FOA) was developed by Pan [25]. In the canonical FOA, each fruit fly is the basic component of the fly population. Through two phases of search, i.e., smell-based search process and vision-based search process [25–31], each fruit fly can reach the optimal searching space in an acceptable computational duration. To the best of our knowledge, there is no literature on solving the steelmaking casting problem by using the FOA. The steelmaking casting problem is an NP-hard combinatorial optimisation problem with a large search space, which is difficult to solve effectively by traditional methods [16–24]. FOA is applicable to the steelmaking casting problem considered in this study for following reasons: first, FOA has only a few parameters, which makes it easy to be implemented; second, the application of FOA in financial distress [25], PID controller tuning [26], general regression neural network optimisation [27], multidimensional knapsack problem [28], web auction logistics service [29], semiconductor final testing scheduling problem [30], and fractional order fuzzy-PID controller for electronic throttle [31], has verified that FOA is applicable for solving many types of scheduling problems and is also competitive to other optimisation algorithms; third, similar to other intelligent algorithms, such as genetic algorithm (GA) and PSO, FOA is an evolutionary algorithm with a parallel search framework in which many heuristics, meta-heuristics and operators can be embedded. In addition, problem-dependent local search approaches can also be easily incorporated into the search framework of the FOA to further enhance exploitation ability, which makes it easy to apply to solve real-world scheduling problems. Therefore, in this study, we propose an improved FOA to solve the realistic HFS in steelmaking casting industries.

To the best of our knowledge, this is the first research work aims to develop an effective fruit fly optimisation algorithm to solve the steelmaking casting problem. The main contributions of this study are as follows: (1) to enhance the balance of exploitation and exploration ability, we design four neighbourhood structures and a self-adaptive neighbourhood strategy; (2) to induce the whole population to move to promising search spaces, we embed well-designed smell-based search and vision-based search procedures in the proposed algorithm; and (3) to further improve the exploration ability of the proposed algorithm, we develop an efficient exploration procedure. The rest of this paper is organised as follows: Section 2 briefly describes the problem. Next, the canonical FOA is presented in Section 3. Section 4 presents the framework of the proposed algorithm. Section 5 illustrates the experimental results and compares to the presented performing algorithms from the literature to demonstrate the superiority of the proposed algorithm. Finally, the last section gives the concluding remarks and future research directions.

2. Problem descriptions

Steelmaking process is a complex manufacturing operation in modern iron and steel industries, which basically consists of three consecutive stages, i.e., steelmaking, refining, and continuous casting, The main processes are as follows [4,24]: first, molten iron from iron-making process enters a converter or electric arc furnace, to reduce the undesired impurity contents. In this stage, a group of molten iron is usually called a charge, which is processed in the same converter or electric arc furnace. Second, the charge of molten steel is transferred into a refining furnace, which is called refining stage, where the impurity is further eliminated and the required alloy ingredients are added. Third, the molten steel enters the last stage, that is, the casting stage, where the liquid steel follows down from the tundish and enters a crystalliser. Finally, the steel is solidified into slabs. The realistic constraint is that a set of charges must be continuously processed in the same cast. To formulate the problem model, we usually have the following assumptions:

- There are many sub-stages in the refining stage, which makes the steelmaking an n-stage problem.
- There are several identical parallel machines in each stage, which can be selected by any charge processed through the stage.
- All charges or jobs follow the same processing sequence, that is, from the first stage to the last stage.
- In the last stage, a set of jobs are grouped into a pre-defined cast to be continuously processed in the same caster, which should not be interrupted.
- In the last stage, the setup time of a new cast is considered.
- Transfer times between stages are also considered.
- Each charge should flow through each stage, and select exactly one machine in each stage.
- The processing time for each charge is pre-defined, deterministic, and uninterrupted.

In this study, we model the multi-stage steelmaking scheduling problem as an HFS problem with continuously casting in the last stage. Similar to Refs. [4,24], the objective is to minimise the average sojourn time and the earliness/tardiness penalty, where the sojourn time of a job is the duration between the completion time in first stage and the starting time in the last stage. The detailed mathematical model for the considered problem can be referred in [4].

3. The canonical FOA

The fruit fly optimisation algorithm (FOA) is a new heuristic for global optimisation that mimics the food finding behaviour of the fruit fly [25]. Osphresis and vision are the two characteristics that make the fruit fly superior to other species. The osphresis organs of fruit flies can help them find all types of scents floating in the air, while the vision organs help them find the food source after getting close to the food location. The main steps of the canonical FOA are given as follows [25–31].

Step 1. Randomly initialise the positions for a swarm of fruit flies.

Step 2. For each fruit fly, perform steps 3 and 4.
Step 3. Change the current position at a random direction with a random strength.
Step 4. Estimate the distance value between its current position and the possible food source, and then compute the judged value of smell concentration (S_i), which is the inverse of the distance.
Step 5. Evaluate the fitness value for each fruit fly, and then select the best one with the optimal value as the current best fruit fly and record the best position.
Step 6. Induce the whole swarm of fruit flies to fly to the best position.
Step 7. If the stop condition is satisfied, stop the algorithm; otherwise, go back to step 2.

4. The proposed algorithm

In this section, we provide the detailed implementation of the proposed FOA algorithm, which includes the encoding and decoding, the improved search operators, and the framework of the proposed algorithm.

4.1. Encoding and decoding

Similar to Refs. [4,6], in the proposed algorithm, a permutation-based representation is used. That is, each solution is represented by a string of integers. Each integer in the string corresponds to a charge number. Thus, the length of the string equals to n. Consider an example problem with three stages and sixteen jobs. There are seven machines in the production system. Each stage contains two or three identical parallel machines, which have the same performance capability for the same charge. The processing time of each operation at each stage is given in Table 1. Suppose one solution is represented by {0, 9, 1, 10, 2, 11, 3, 4, 12, 5, 13, 14, 6, 15, 7, 8}, which means that at the first stage, the scheduling sequence is J_0, J_9, J_1, J_{10}, J_2, J_{11}, J_3, J_4, J_{12}, J_5, J_{13}, J_{14}, J_6, J_{15}, J_7, and J_8.

It is obvious that the above solution representation contains no information about the machine assignment. In the proposed algorithm, each job, when arriving at each stage except the last stage, will be assigned to the first available machine in the current stage. At the last stage, for considering the continuously casting condition, each charge to be included in the given cast must be processed on the predefined machine. Therefore, in the proposed algorithm, the decoding steps are given as follows:

Step 1. At the first stage, perform the following steps: (1) select each charge one by one according to the sequence in the solution representation; (2) assign each selected charge to the first available machine; (3) schedule each charge on the assigned machine.
Step 2. At the next stages except the last stage, when each charge completes its task in the previous stage and is transferred to the current stage, the first available machine will be selected to process it.
Step 3. At the last stage, each charge is selected into the predefined cast, and should be processed immediately or later to adapt to the continuous casting.

The Gantt chart of one solution for the above example problem is shown in Fig. 1. In the Gantt chart, each charge is represented by a rectangle and numbered with the charge number. Each stage is divided by a line. At the last stage, each charge should be continuously processed in its assigned cast on the predetermined caster. For example, charge O_3, O_4, and O_5 are processed in the same cast on M_6.

4.2. Neighbourhood structures

In this study, considering the problem structure and the balance of the exploration and exploitation ability, four neighbourhood structures are proposed, which are given as follows:

- Single-swap structure, denoted by N_1. (1) Randomly select two charge numbers in the solution representation; (2) swap the two selected charges in the scheduling string.
- Insert structure, denoted by N_2. (1) Randomly select two positions r_1 and r_2 in the solution representation, where $r_1 < r_2$; (2) insert the charge at the position r_2 before r_1 in the scheduling string.
- Multi-swap structure, denoted by N_3. Similar to [4], perform the single-swap structure several times.
- Multi-insert structure, denoted by N_4. (1) Randomly generate a limit number h between h_1 and h_2, where h_1 and h_2 are the lower and upper bound of the loop number and are experimentally set to 5 and 10, respectively; (2) perform the following steps h times: randomly select one position r in the solution representation, insert the element at the positions $r+1$ before the position $r-1$. If $r=0$, set $r-1$ equal to n; if $r=n$, set $r+1$ with 0.

4.3. A self-adaptive neighbourhood strategy

The neighbouring approaches, introduced in Section 4.2, have different roles for the convergence capability or the population diversity. To balance the exploration and exploitation capability during the evolution, that is, to enhance the search ability while holding population diversity and utilise different neighbourhood structure in different stages, we introduce a self-adaptive strategy, which is similar to [32,33]. The detailed steps of the proposed self-adaptive strategy are given as follows.

Step 1. Set the function parameter, such as the length of the neighbourhood vector N_s, and the refill probability R_p.
Step 2. Initialise a neighbourhood vector (named NV), with length equals to N_s, by filling with a random neighbourhood structure taken from the structures discussed in Section 4.2.
Step 3. Generate an empty wining neighbourhood vector (named WNV), with length equals to N_s.
Step 4. After receiving a call for selection of a neighbourhood structure, perform Steps 5 to 7.
Step 5. If NV is not empty, take the first neighbouring structure from NV to generate a neighbouring solution of the current one. If the new neighbouring solution is better than the current solution, replace the latter with the former,

Table 1
Processing times.

	J_0	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}	J_{11}	J_{12}	J_{13}	J_{14}	J_{15}
Stage1	47	41	46	46	39	41	41	38	38	49	43	38	47	45	45	42
Stage2	38	41	42	37	42	49	36	48	42	44	43	37	45	41	50	40
Stage3	42	48	37	43	41	37	38	37	48	44	40	39	40	49	42	40

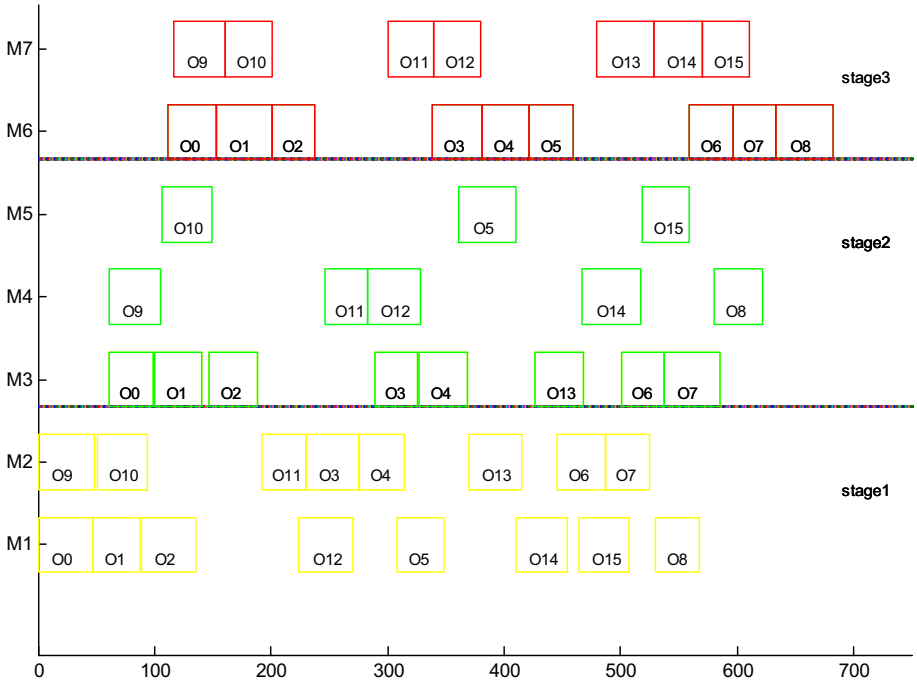

Fig. 1. Gantt chart for the example problem.

and insert the corresponding neighbourhood structure into WNV.

Step 6. If NV is empty and WNV is not empty, fill NV with the elements of the current WNV. If the length of the new NV is less than N_s, the empty positions will be filled as follows: 75% is refilled from the WNV, and the remaining 25% is refilled by a random selection from the four neighbourhood structures explained in Section 4.2.

Step 7. If WNV is empty, the new NV will be filled as follows: 50% from the latest NV, and the remaining 50% is randomly selected from the neighbourhood structures.

4.4. Population initialisation

The FOA algorithm begins with a population of *PS* initial fruit flies. A population with a high level of solution quality and diversity is very crucial for the algorithm. In this study, we propose an initialisation heuristic for steelmaking problem, which is given as follows:

Step 1. Randomly generate a solution.
Step 2. Schedule each charge through the first stage to the last stage.
Step 3. Sequence the resulting starting time for each charge determined from the last step in a non-decreasing order. Next, we obtain the solution representation that is the same as the charge sequence in the order.

Using the above initialisation heuristic, we initialise the population as follows:

Step 1. Generate a solution by using the above heuristic.
Step 2. Let counter $Cnt = 1$, perform the following steps until $Cnt = PS$.
Step 3. Generate a solution in a random way and evaluate it. If the newly-generated solution is not the same as any individual in the current population, insert it into the population and let $Cnt = Cnt + 1$; otherwise, discard it.
Step 4. Go back to step 3.

4.5. Smell-based search procedure

In FOA, each fruit fly utilises the smell-based search procedure to perform the exploitation task, which is implemented as follows.

Step 1. For each fruit fly S_i, perform the following steps for *SN* times.
Step 2. Generate a neighbouring fruit fly around S_i by using the self-adaptive neighbourhood strategy discussed in Section 4.3.
Step 3. Evaluate the newly-generated neighbouring solution.
Step 4. If the newly-generated neighbouring solution is better than the best solution found so far, then replace the best solution with the neighbouring solution; if the neighbouring one is better than the current solution, and then replace the latter with the former.
Step 5. Update the two neighbouring vectors NV and WNV.

4.6. Vision-based search procedure

In the canonical FOA, the vision-based search procedure is used to induce the entire population to move to a better searching space; therefore the whole swarm will improve their searching ability. However, the vision-based search procedure should consider the balance of exploitation and exploration ability. If the whole swarm flies to the same searching space, the population will prematurely converge to local optima. In this study, we propose an improved vision-based search procedure, which can both improve the performance of the current population and retain the diversity of the whole swarm. The detailed steps are given as follows.

Step 1. For the entire population, performs steps 2 to 3.

Step 2. Sort the entire population in a non-decreasing order according to their fitness values.
Step 3. Find the worst fruit fly in the current population, and then replace it with the best fruit fly found thus far.

4.7. Exploration procedure

For solving large scale problems, the exploration ability of the proposed algorithm is critical. If the algorithm lacks exploration ability, the entire population will converge to a worse result and a premature solution occurs. To avoid sticking to local optima, in this study, we propose an exploration procedure in the proposed FOA, which is given as follows.

Step 1. For each fruit fly S_i, record the update iteration number in a vector.
Step 2. In each generation, set the update iteration number for each fruit fly as follows: if the fruit fly is updated by a newly-generated neighbouring individual, set the update iteration number to zero; otherwise, increase it.
Step 3. Sort each fruit fly in the current population according to their update iteration numbers.
Step 4. Find the fruit fly with the update iteration number greater than L_n and denote it as M_i. If there is more than one satisfactory fruit fly, then randomly select one.
Step 5. For the best fruit fly found so far, perform the smell-search procedure ten times by using the neighbourhood discussed in Section 4.2. Replace M_i with the newly-generated fruit fly.

4.8. The framework of the FOA algorithm

The detailed steps of the proposed FOA algorithm are as follows:

Step 1. Initialisation phase.
Step 1.1. Set the system parameters.
Step 1.2. Initialise the population.
Step 2. Evaluate each fruit fly in the population.
Step 3. If the stopping condition is satisfied, stop the algorithm; otherwise, perform steps 4 to 7.
Step 4. Perform the smell-based search procedure discussed in sub-Section 4.5.
Step 5. Perform the vision-based search procedure discussed in sub-Section 4.6.
Step 6. Perform the exploration procedure discussed in sub-Section 4.7.
Step 7. Go back to step 3.

From the proposed framework, the main search procedures are realised through steps 4 to 6. Step 4 is the smell-based search procedure, which performs the exploitation tasks for each fruit fly. The vision-based search procedure is performed in step 5, which induces the entire population towards a better searching space. The exploration task is completed through the exploration procedure. The proposed self-adaptive neighbourhood strategy further enhances the balance of exploration and exploitation ability of FOA, which makes the proposed algorithm applicable to solve the steelmaking casting problem with a huge search space.

4.9. Computational complexity analysis

From the framework, we can see that the proposed FOA algorithm contains three main procedures. Therefore, similar to reference [30], we also provide a computational complexity analysis from the three procedures: (1) for the smell-based search procedure, each fruit fly generates and evaluates SN neighbouring solutions to perform the exploitation task. Therefore, the computational complexity of the smell-based procedure is $O(PS \times SN \times nms)$, where n, m, and s represent the total number of jobs, machines, and stages, respectively; (2) for the vision-based search procedure, the computational complexity is $O(PS \ln PS)$ to find the worst solution in the current population. Therefore, the computational complexity of the vision-based procedure is approximately $O(PS \ln PS)$; (3) for the exploration procedure, each fruit fly should record the update iteration number, which consumes $O(PS)$. In addition, to sort the whole population consumes $O(PS \ln PS)$. Meanwhile, the newly-generated fruit fly takes $O(nms)$. Therefore, the computation complexity of the exploration procedure is approximately $O(nms)$.

From the above computational complexity analysis, we can conclude that the proposed FOA is acceptable and that the algorithm could solve the real-world steelmaking casting problems efficiently.

5. Numerical results

This section discusses the computational experiments used to evaluate the performance of the proposed algorithm. Our algorithm was implemented in C++ on an Intel Core i7 3.4 GHz PC with 16 GB of memory. The compared algorithms include hGA [34], GAS [35], TSSCS [36], and ABC [4]. To make a fair comparison for solving the realistic steelmaking casting problems, we coded the above algorithms and adopted the parameter settings proposed in their references, respectively, except the computational times for each instance are set to 20 s. The best, worst, and average results of experiments for the above 20 problems from 20 independent runs were collected for performance comparisons. The performance measure is relative percentage increase (RPI), which is calculated as follows:

$$\mathrm{RPI(C)} = \frac{C_c C_b}{C_b} \times 100 \tag{1}$$

where C_b is the best solution found by all compared algorithms, and C_c is the best solution collected by a given compared algorithm.

5.1. Experimental instances

In this study, we generate twenty problem instances, named from "Case1" to "Case20", according to the practical situations of the iron and steel production in a Baosteel complex, the largest and most advanced iron and steel enterprise in China. The detailed parameters are as follows:

- There are three main stages in the shop, i.e., the steelmaking stage, the refining stage, and the continuous casting stage. To make the problem closer to the realistic production, we divide the refining stage into 1–3 sub-stages. Therefore, we obtain 3–6 stages in the shop.
- There are 5–6 parallel converts in the steelmaking stage, 5–6 identical refining furnaces in the refining stage, and 5–6 continuous casters in the last stage.
- Each continuous caster processes 3–4 casts in each workday, and each cast generally contains 2–6 jobs, which should be processed continuously. That is, we consider a total of 15–24 casts and a total of 120 charges or so in the shop.
- For each charge or job, the processing times in the steelmaking, refining, and casting are in the ranges of [36–50].
- For each machine, the release time is not considered as a technical capability.

Table 2
Combinations of the parameter values.

Parameter	Level			
	1	2	3	4
PS	10	30	50	100
SN	1	3	5	10

- The transfer times for each two consecutive stages are range in [10–15].
- The setup time for each cast is set to 100.
- The predefined starting time of the first cast on each caster in the continuous stage can be estimated by the sum of the processing time and the transfer time of each charge related to the cast.

5.2. Experimental parameters

The parameters for the self-adaptive neighbourhood strategy are set according to the reference [33]: (1) The length of the neighbourhood vector N_s is set to 10; (2) The refill probability R_p is set to 0.75.

Similar to Ref. [33], the number of iterations during which the solution does not improve (L_n) is also set to 20. The remaining two parameters are the population size (*PS*) and the size of neighbouring in the smell-based search process (*SN*) for the FOA procedure. The levels of the two parameters are given in Table 2. The Taguchi method of DOE [37] is utilised to test the influence of these two parameters on the performance of the proposed algorithm. The tested instance is Case20. Because the two parameters are set with four factor levels, an orthogonal array $L_{16}(4^2)$ is selected. For each parameter combination, the proposed algorithm is run 20 times independently, and then the average makespan value obtained by the proposed algorithm is collected as the response variable (RV). Table 3 presents the response values of different combinations of these two parameters. Fig. 2 shows the factor level trend of the two parameters.

It can be seen from Fig. 2 that the proposed algorithm exhibits a better performance under the two parameters with following levels: *PS* with level 1 and *SN* with level 2. From Fig. 2, we can see that the parameter *SN* is more critical than *PS* in the proposed algorithm. A large value of *SN* means more computational resources consumed in the smell-based search procedure, causing the algorithm to lose the exploration ability. An excessively small value of *SN* means the loss of exploitation ability of the algorithm. Therefore, to balance the exploration and exploitation ability, the suitable value for the key parameter *SN* is set to 3 in the proposed FOA. According to the above analysis, the suitable values for the two considered parameters are set to 10 and 3 for *PS* and *SN*, respectively.

Table 3
Orthogonal array and RV values.

Experiment number	Factor		RV
	PS	SN	
1	1	1	2492.02
2	1	2	2475.52
3	1	3	2511.98
4	1	4	2486.23
5	2	1	2494.05
6	2	2	2487.8
7	2	3	2491.07
8	2	4	2502.68
9	3	1	2505.16
10	3	2	2485.11
11	3	3	2496.95
12	3	4	2486.73
13	4	1	2493.93
14	4	2	2484.05
15	4	3	2495.66
16	4	4	2497.52

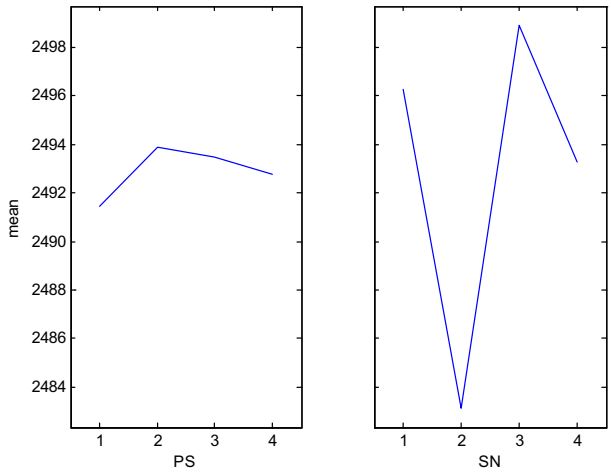

Fig. 2. Factor level trend.

Table 4
Comparison results of the RPI values for the best makespan (the minimum RPI values are in bold).

Problem	Scale	hGA	GAS	TSSCS	ABC	FOA
Case1	3-62-16	8.13	2.32	1.30	**0.00**	0.77
Case2	3-58-17	1.76	2.54	3.02	**0.00**	0.08
Case3	3-68-16	**0.00**	1.54	2.15	0.27	0.87
Case4	3-90-15	3.58	3.45	2.55	0.39	**0.00**
Case5	3-84-18	5.74	4.42	6.13	7.51	**0.00**
Case6	4-79-23	0.23	0.47	1.10	**0.00**	0.15
Case7	4-67-23	2.98	0.81	1.03	0.41	**0.00**
Case8	4-70-22	1.77	2.38	2.64	1.82	**0.00**
Case9	4-104-22	8.26	3.67	2.73	2.86	**0.00**
Case10	4-77-22	4.07	2.70	2.18	3.04	**0.00**
Case11	5-78-28	0.76	0.10	**0.00**	0.51	0.21
Case12	5-91-27	1.56	0.69	0.79	0.89	**0.00**
Case13	5-73-26	1.76	0.01	0.77	0.77	**0.00**
Case14	5-72-26	1.72	1.46	1.74	**0.00**	0.26
Case15	5-91-28	2.08	0.47	**0.00**	1.02	1.39
Case16	6-99-33	1.92	**0.00**	2.81	1.86	1.22
Case17	6-103-32	0.38	**0.00**	0.41	0.36	0.03
Case18	6-98-34	0.72	0.41	0.15	**0.00**	0.12
Case19	6-79-33	0.65	0.71	1.37	0.19	**0.00**
Case20	6-88-33	3.91	0.21	0.65	0.92	**0.00**
Average		2.60	1.42	1.68	1.14	**0.26**

5.3. Comparisons analysis

To solve the above twenty realistic steelmaking problems, we implemented the four presented algorithms: hGA, GAS, TSSCS, and ABC. Note that hGA is an efficient algorithm for solving HFS problems, while GAS and TSSCS are the two algorithms considering HFS with limited buffers, and ABC is the very recently presented algorithm for solving the steelmaking scheduling problem. The above four compared algorithms are efficient for solving HFS. For each compared algorithm, we take 20 independent runs for each instance. After each independent run, the best, worst, and average values are collected to make detailed comparisons.

Table 4 lists the comparison results of the RPI values for the best makespan obtained by each compared algorithm for each considered instance. There are seven columns in the table. The first column presents the instance name. The following column presents the scale of the problem, in which the three numbers correspond to the number of stages, charges, and parallel machines. The results of the best RPI values obtained by the compared algorithms,

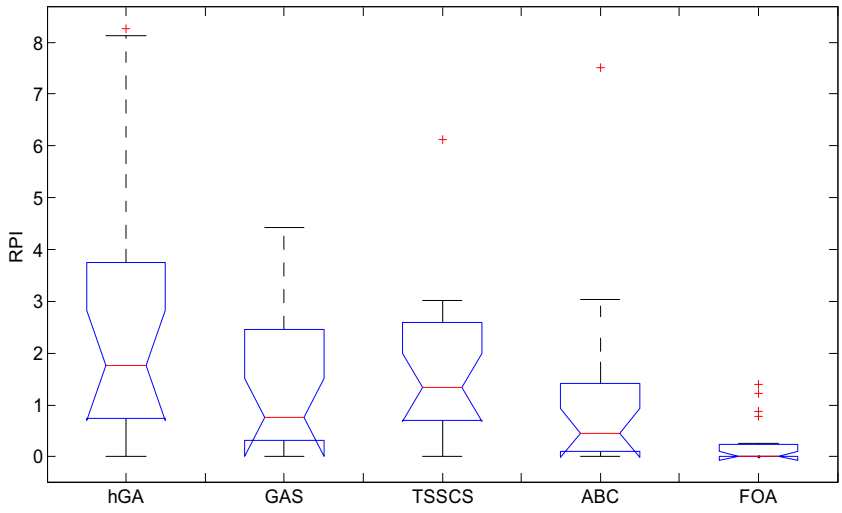

Fig. 3. Means and 95% LSD interval for the best values of the five compared algorithms (p-value = 0.0005).

Table 5
Comparison results of the RPI values for the worst makespan (the minimum RPI values are in bold).

Problem	Scale	hGA	GAS	TSSCS	ABC	FOA
Case1	3-62-16	27.66	22.10	25.28	27.44	**14.34**
Case2	3-58-17	19.78	11.08	11.07	14.79	**10.79**
Case3	3-68-16	19.21	8.46	15.31	10.68	**6.57**
Case4	3-90-15	17.31	18.09	18.31	17.02	**8.67**
Case5	3-84-18	26.54	**14.30**	28.27	24.08	15.23
Case6	4-79-23	5.42	4.22	5.60	4.37	3.82
Case7	4-67-23	6.35	5.97	8.04	5.54	2.76
Case8	4-70-22	13.98	11.74	10.07	**4.55**	6.95
Case9	4-104-22	21.99	**9.93**	12.18	10.85	10.98
Case10	4-77-22	10.09	14.16	11.15	9.58	**8.31**
Case11	5-78-28	4.89	5.10	6.82	**3.79**	5.51
Case12	5-91-27	8.59	9.18	9.46	**5.07**	6.28
Case13	5-73-26	7.20	7.07	4.36	7.70	2.45
Case14	5-72-26	9.01	**6.55**	8.76	7.52	7.15
Case15	5-91-28	11.52	5.17	10.09	5.61	**4.86**
Case16	6-99-33	8.80	8.06	8.95	**4.38**	5.15
Case17	6-103-32	2.87	4.10	6.20	**2.09**	2.45
Case18	6-98-34	3.80	2.30	2.47	**1.45**	1.49
Case19	6-79-33	4.82	4.03	5.02	**3.32**	3.78
Case20	6-88-33	8.24	**4.13**	6.75	5.72	4.31
Average		11.90	8.79	10.71	8.78	**6.59**

Table 6
Comparison results of the RPI values for the average makespan (the minimum RPI values are in bold).

Problem	Scale	hGA	GAS	TSSCS	ABC	FOA
Case1	3-62-16	16.15	9.01	11.15	8.81	**4.97**
Case2	3-58-17	7.86	6.53	6.53	5.13	**4.95**
Case3	3-68-16	10.24	4.83	7.90	4.71	**3.67**
Case4	3-90-15	9.44	9.81	9.23	4.00	**3.21**
Case5	3-84-18	13.53	8.40	13.08	15.94	**6.13**
Case6	4-79-23	2.22	2.35	3.00	1.70	**1.55**
Case7	4-67-23	4.82	2.87	4.20	2.16	**1.69**
Case8	4-70-22	5.97	5.05	5.73	**3.27**	**3.69**
Case9	4-104-22	12.61	6.95	6.37	6.80	**3.37**
Case10	4-77-22	7.56	6.68	6.69	5.76	**4.08**
Case11	5-78-28	3.18	2.38	4.70	**1.59**	1.77
Case12	5-91-27	4.34	3.26	4.44	**2.45**	2.59
Case13	5-73-26	3.93	2.66	2.49	2.64	**0.91**
Case14	5-72-26	5.20	4.29	4.82	3.80	**3.39**
Case15	5-91-28	6.74	3.00	4.33	3.08	**2.68**
Case16	6-99-33	5.28	4.10	4.47	2.97	**2.96**
Case17	6-103-32	1.62	1.88	2.09	1.02	**0.69**
Case18	6-98-34	1.91	1.06	1.44	0.86	**0.62**
Case19	6-79-33	2.95	2.60	3.17	**1.61**	1.73
Case20	6-88-33	5.65	**2.01**	2.65	3.18	**2.01**
Average		6.56	4.49	5.42	4.07	**2.83**

Table 7
Comparison results of the computational times (time unit: seconds, the minimum values are in bold).

Problem	Scale	hGA	GAS	TSSCS	ABC	FOA
Case1	3-62-16	8.69	12.12	12.06	12.14	**8.07**
Case2	3-58-17	**6.85**	7.75	14.65	12.37	8.69
Case3	3-68-16	12.82	9.30	9.53	14.02	**8.64**
Case4	3-90-15	11.77	14.78	13.52	14.81	**10.08**
Case5	3-84-18	11.46	14.03	13.95	17.05	**9.43**
Case6	4-79-23	7.83	10.07	**6.98**	15.50	10.74
Case7	4-67-23	8.73	**8.23**	9.95	12.87	9.49
Case8	4-70-22	**7.32**	12.15	12.10	12.60	10.90
Case9	4-104-22	12.02	17.89	**11.51**	17.03	14.99
Case10	4-77-22	**8.43**	13.30	12.37	14.59	9.89
Case11	5-78-28	12.88	10.83	11.47	17.63	**9.44**
Case12	5-91-27	13.58	14.15	**12.11**	17.53	12.95
Case13	5-73-26	**8.21**	10.61	9.49	11.50	11.07
Case14	5-72-26	10.61	13.12	**9.27**	15.10	9.72
Case15	5-91-28	12.22	15.36	**10.46**	14.43	11.24
Case16	6-99-33	12.47	13.29	**8.64**	17.80	9.26
Case17	6-103-32	16.16	18.22	**10.04**	16.88	10.43
Case18	6-98-34	15.79	18.43	7.84	17.62	**7.55**
Case19	6-79-33	**9.04**	15.00	10.72	15.87	10.06
Case20	6-88-33	15.55	16.29	**9.87**	16.16	12.05
Average		11.12	13.25	10.83	15.17	**10.23**

including hGA, GAS, TSSCS, ABC, and FOA, are listed in the following five columns. The last row in the table presents the average value of the best RPI values obtained by each compared algorithm for all of the twenty instances. It can be concluded from Table 4 that: (1) the FOA obtained ten best values out of twenty instances, which is obviously better than the other four compared algorithms; (2) the last row shows that the proposed FOA algorithm obtained an average value with 0.26 for all of the twenty instances, which is obviously better than the other four compared algorithms. The following algorithms are ABC, GAS, TSSCS, and hGA.

To check whether the observed differences from the above table is indeed significantly different, we also performed a multifactor analysis of variance (ANOVA), where the five compared algorithms are considered as factors. This method has been used to compare different algorithms in [4,28,30], and among many others. Fig. 3 illustrates the means and the 95% LSD interval for the best values of the five compared algorithms. In Fig. 3, a p-value close to zero suggests that there is a statistically significant difference between the proposed algorithm and the other four compared algorithms. It can be concluded from Fig. 3 that the proposed FOA is significantly better than the other four compared algorithms for the considered problems.

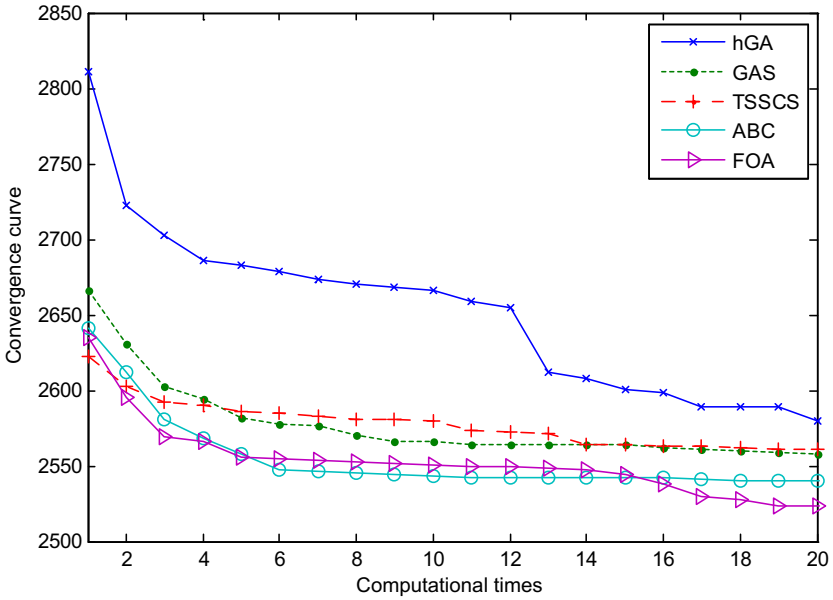

Fig. 4. Comparisons of the convergence curve for Case20.

Tables 5 and 6 present the comparison results of the RPI values for the worst and average makespan obtained by the five compared algorithms, respectively. From the two comparison tables, we can conclude that: (1) in the worst situation, the proposed algorithm obtains nine optimal values out of twenty instances; (2) in the average situation, the FOA algorithm obtains sixteen optimal values out of twenty instances; (3) the average values for the worst and average situations indicate that the proposed algorithm is better than the other four compared algorithms, which verifies the efficiency and robustness of the proposed algorithm.

Table 7 presents the comparisons of the computational times consumed by the five compared algorithms for solving the given instances. It can be seen from Table 7 that the proposed FOA required the least average computational times among the five compared algorithms. The following algorithms are TSSCS, hGA, GAS, and ABC. Fig. 4 gives the comparisons of the convergence curve for solving Case20. It can be concluded from Fig. 4 that the proposed algorithm exhibits perfect convergence ability for solving the large scale realistic HFS problem. Considering both the performance and the computational times, it can be concluded that the proposed FOA is superior to the other four compared algorithms for solving steelmaking casting problems with different scales.

6. Conclusion

In this study, an effective fruit fly optimisation is proposed to solve the realistic steelmaking casting problems. The main contributions and innovations of the proposed FOA are as follows:

(1) This is the first research work on the application of the FOA algorithm to solve the steelmaking casting problem.
(2) A self-adaptive neighbourhood strategy is embedded in the proposed algorithm, which can enhance the balance of exploitation and exploration ability.
(3) Well-designed smell-based search and vision-based search procedures are embedded in the framework of FOA, which can induce the whole population to move to promising search spaces.
(4) An exploration procedure is utilised in the proposed FOA, which can further improve the exploration ability of the proposed algorithm.

This research work not only provides a powerful optimisation algorithm for the HFS in steelmaking casting system, but it also enriches the application of the FOA in real-world scheduling field. In the proposed FOA framework, there exist only a few control parameters; as a result it easy to embed some knowledge-based rules and other heuristic or meta-heuristic strategies. From the experimental results and analysis, it can be concluded that the proposed FOA is suitable for solving the real-world scheduling problems in a static environment. However, in most realistic production system, many disruption events will cause the solution to be infeasible. Moreover, multi-objective constraints should also be considered in the future work of FOA applications. Therefore, the future work is to apply the proposed algorithm for solving other potential realistic applications under multi-objective constraints and dynamic environments.

Acknowledgments

This research is partially supported by National Science Foundation of China 61174187, 51435009 and 61104179, Program for New Century Excellent Talents in University (NCET-13-0106), Specialized Research Fund for the Doctoral Program of Higher Education (20130042110035), Science Foundation of Liaoning Province in China (2013020016), Basic scientific research foundation of Northeast University under Grant N110208001 and N130508001, Starting foundation of Northeast University under Grant 29321006, and IAPI Fundamental Research Funds (2013ZCX02).

References

[1] J.N.D. Gupta, Two-stage, hybrid flow shop scheduling problem, J. Oper. Res. Soc. 39 (1988) 359–364.
[2] R. Ruiz, J.A. Vázquez Rodríguez, The hybrid flow shop scheduling problem, Eur. J. Oper. Res. 205 (2010) 1–18.
[3] I. Ribas, R. Leisten, J.M. Framinan, Review and classification of hybrid flow shop scheduling problems from a production systems and a solutions procedure perspective, Comput. Oper. Res. 37 (2010) 1439–1454.
[4] Q.K. Pan, L. Wang, K. Mao, J.H. Zhao, M. Zhang, An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process, IEEE Trans. Autom. Sci. Eng. 10 (2) (2013) 307–322.
[5] S. Voss, A. Witt, Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: a real-world application, Int. J. Prod. Econ. 105 (2) (2007) 445–458.

[6] R. Ruiz, F.S. Şerifoğlu, T. Urlings, Modeling realistic hybrid flexible flowshop scheduling problems, Comput. Oper. Res. 35 (4) (2008) 1151–1175.
[7] J. Behnamian, S.M.T. Fatemi Ghomi, M. Zandieh, A multi-phase covering Pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheuristic, Expert Syst. Appl. 36 (8) (2009) 11057–11069.
[8] F. Dugardin, F. Yalaoui, L. Amodeo, New multi-objective method to solve reentrant hybrid flow shop scheduling problem, Eur. J. Oper. Res. 203 (1) (2010) 22–31.
[9] A. Costa, F.A. Cappadonna, S. Fichera, A dual encoding-based meta-heuristic algorithm for solving a constrained hybrid flow shop scheduling problem, Comput. Ind. Eng. 64 (4) (2013) 937–958.
[10] E. Figielska, A heuristic for scheduling in a two-stage hybrid flowshop with renewable resources shared among the stages, Eur. J. Oper. Res. 236 (2) (2014) 433–444.
[11] H. Luo, A. Zhang, G.Q. Huang, Active scheduling for hybrid flowshop with family setup time and inconsistent family formation, J. Intell. Manuf., doi: 10.1007/s10845-013-0771-9.
[12] L.C. Wang, Y.Y. Chen, T.L. Chen, C.Y. Cheng, C.W. Chang, A hybrid flowshop scheduling model considering dedicated machines and lot-splitting for the solar cell industry, Int. J. Syst. Sci., doi: 10.1080/00207721.2012.762557.
[13] T.P. Chung, C.J. Liao, An immunoglobulin-based artificial immune system for solving the hybrid flow shop problem, Appl. Soft Comput. 13 (8) (2013) 3729–3736.
[14] F.D. Chou, Particle swarm optimisation with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessor tasks, Int. J. Prod. Econ. 141 (1) (2013) 137–145.
[15] J. Yang, A two-stage hybrid flow shop with dedicated machines at the first stage, Comput. Oper. Res. 40 (12) (2013) 2836–2843.
[16] L. Tang, P.B. Luh, J. Liu, L. Fang, Steel-making process scheduling using Lagrangian relaxation, Int. J. Prod. Res. 40 (1) (2002) 55–70.
[17] H. Xuan, L.X. Tang, Scheduling a hybrid flowshop with batch production at the last stage, Comput. Oper. Res. 34 (9) (2007) 2718–2733.
[18] L.X. Tang, J.Y. Liu, A.Y. Rong, Z.H. Yang, A mathematical programming model for scheduling steelmaking-continuous casting production, Eur. J. Oper. Res. 120 (2) (2000) 423–435.
[19] A. Atighehchian, M. Bijari, H. Tarkesh, A novel hybrid algorithm for scheduling steel-making continuous casting production, Comput. Oper. Res. 36 (8) (2009) 250–2461.
[20] H. Missbauer, W. Hauber, W. Stadler, A scheduling system for the steelmaking-continuous casting process. A case study from the steel-making industry, Int. J. Prod. Res. 47 (15) (2009) 4147–4172.
[21] Y.Y. Tan, S.X. Liu, Models and optimisation approaches for scheduling steelmaking-refining-continuous casting production under variable electricity price, Int. J. Prod. Res. 52 (4) (2014) 1032–1049.
[22] L.X. Tang, J.X. Luo, J.Y. Liu, Modelling and a tabu search solution for the slab reallocation problem in the steel industry, Int. J. Prod. Res. 51 (14) (2013) 4405–4420.
[23] H. Xuan, B. Li, Scheduling dynamic hybrid flowshop with serial batching machines, J. Oper. Res. Soc. 64 (2013) 825–832.
[24] K. Mao, Q.K. Pan, X. Pang, T. Chai, A novel Lagrangian relaxation approach for a hybrid flowshop scheduling problem in the steelmaking-continuous casting process, Eur. J. Oper. Res. 236 (1) (2014) 51–60.
[25] W.T. Pan, A new fruit fly optimisation algorithm: taking the financial distress model as an example, Knowl.-Based Syst. 26 (2012) 69–74.
[26] J. Han, P. Wang, X. Yang, Tuning of PID controller based on fruit fly optimisation algorithm, Int. Conf. Mechatron. Autom. (ICMA) (2012) 409–413.
[27] H. Li, S. Guo, C. Li, J. Sun, A hybrid annual power load forecasting model based on generalized regression neural network with fruit fly optimisation algorithm, Knowl.-Based Syst. 37 (2013) 378–387.
[28] L. Wang, X.L. Zheng, S.Y. Wang, A novel binary fruit fly optimisation algorithm for solving the multidimensional knapsack problem, Knowl.-Based Syst. 48 (2013) 17–23.
[29] S.M. Lin, Analysis of service satisfaction in web auction logistics service using a combination of Fruit fly optimisation algorithm and general regression neural network, Neural Comput. Appl. 22 (3–4) (2013) 783–791.
[30] X.L. Zheng, L. Wang, S.Y. Wang, A novel fruit fly optimisation algorithm for the semiconductor final testing scheduling problem, Knowl.-Based Syst. 57 (2014) 95–103.
[31] W. Sheng, Y. Bao, Fruit fly optimisation algorithm based fractional order fuzzy-PID controller for electronic throttle, Nonlinear Dyn. 73 (1–2) (2013) 611–619.
[32] J.Q. Li, Q.K. Pan, Chemical-reaction optimisation for flexible job-shop scheduling problems with maintenance activity, Appl. Soft Comput. 12 (9) (2012) 2896–2912.
[33] Q.K. Pan, M. Tasgetiren, P.N. Suganthan, T.J. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Inf. Sci. 181 (12) (2011) 2455–2468.
[34] R. Ruiz, C. Maroto, A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility, Eur. J. Oper. Res. 169 (2006) 781–800.
[35] V. Yaurima, L. Burtseva, A. Tchernykh, Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers, Comput. Ind. Eng. 56 (4) (2009) 1452–1463.
[36] X. Wang, L. Tang, A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers, Comput. Oper. Res. 36 (3) (2009) 907–918.
[37] D.C. Montgomery, Design and analysis of experiments, John Wiley & Sons, Arizona, 2005.