Information Sciences 316 (2015) 487–502

Contents lists available at ScienceDirect

Information Sciences

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

Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm ☆

Jun-qing Li [a,b], Quan-ke Pan [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] State Key Lab of Digital Manufacturing Equipment & Technology, Huazhong University of Science & Technology, Wuhan 430074, PR China

ARTICLE INFO

Article history:
Received 19 November 2013
Received in revised form 18 September 2014
Accepted 3 October 2014
Available online 13 October 2014

Keywords:
Hybrid flow shop scheduling problem
Artificial bee colony
Tabu search
Limited buffer

ABSTRACT

This paper presents a novel hybrid algorithm (TABC) that combines the artificial bee colony (ABC) and tabu search (TS) to solve the hybrid flow shop (HFS) scheduling problem with limited buffers. The objective is to minimize the maximum completion time. Unlike the original ABC algorithm, in TABC, each food source is represented by a string of job numbers. A novel decoding method is embedded to tackle the limited buffer constraints in the schedules generated. Four neighborhood structures are embedded to balance the exploitation and exploration abilities of the algorithm. A TS-based self-adaptive neighborhood strategy is adopted to impart to the TABC algorithm a learning ability for producing neighboring solutions in different promising regions. Furthermore, a well-designed TS-based local search is developed to enhance the search ability of the employed bees and onlookers. Moreover, the effect of parameter setting is investigated by using the Taguchi method of design of experiment (DOE) to determine the suitable values for key parameters. The proposed TABC algorithm is tested on sets of instances with large scales that are generated based on realistic production. Through a detailed analysis of the experimental results, the highly effective and efficient performance of the proposed TABC algorithm is contrasted with the performance of several algorithms reported in the literature.

1. Introduction

In recent years, the classical flow shop scheduling problem (FSSP) has been proven to play an important role in modern manufacturing and production systems. The hybrid flow shop (HFS) scheduling problem is one branch of the FSSP and has been verified to be an NP-hard problem [11]. Recent and comprehensive reviews on the HFS can be found in [33,36]. Many heuristics for solving the HFS have been developed. Of these heuristic or meta-heuristic algorithms, genetic algorithms (GAs) are the most popular. Portmann et al. introduced an enhanced version of the B&B algorithm crossed with a GA [32]. Jin et al. solved a scheduling problem involving a real printed circuit board manufacturing system using a GA method [13]. Oguz and Ercan developed a GA approach for the HFS with multiprocessor tasks [28]. Ruiz and Maroto developed a GA for the HFS with

☆ This research is partially supported by National Science Foundation of China 61174187, 51435009, 61374187, 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).

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

http://dx.doi.org/10.1016/j.ins.2014.10.009
0020-0255/

sequence-dependent setup times and machine eligibility [34]. Engin et al. developed an efficient GA for the HFS with multiprocessor tasks [6]. Other strategies have also been introduced for solving the HFS. Lin and Liao solved the two-stage HFS problem with setup time and dedicated machines [23]. Babayan and He presented an agent-based approach for a three-stage HFS with identical parallel machines [2]. Ying and Lin solved the problem by using ant colony optimization (ACO) [48]. Zandieh et al. applied artificial immune systems (AIS) for the HFS with sequence-dependent setup times [49]. Janiak et al. applied three constructive algorithms and three meta-heuristics based on the tabu search (TS) and simulated annealing (SA) algorithms [12]. More recently, Kahraman et al. investigated a parallel greedy algorithm for solving the multistage HFS [14]. Liao et al. proposed an approach using particle swarm optimization (PSO) and bottleneck heuristics for the problem [22]. The realistic HFS is a much more complex generalization of the traditional HFS. Ruiz et al. solved realistic HFS problems with skipped stages, sequence-dependent setup times, machine lags, release dates, machine eligibility and precedence relationships [35]. Zandieh and Gholami solved the HFS with stochastic machine breakdown by using an immune algorithm (IA) [50]. Dugardin et al. focused on the multi-objective resolution of a re-entrant HFS [5]. Recently, Xuan and Li investigated the batch decomposition strategy with a mixed backward and forward dynamic programming algorithm [46]. Pan et al. proposed an efficient artificial bee colony (ABC) algorithm for solving the steelmaking problem by considering the assignment of different penalty coefficients for three objectives, i.e., the average sojourn time and the earliness/tardiness penalty [31]. Indeed, the literature reveals that the HFS has become increasingly important in realistic production systems.

The majority of the literature makes the assumption that there are sufficient intermediate buffers between consecutive stages. However, in realistic scheduling problems, a finite intermediate buffer (limited buffer) always exists, which blocks and delays operation on the previous machine or the intermediate buffer. Several heuristic and meta-heuristic algorithms that solve the flow shop scheduling problem with limited buffers have been reported. Brucker et al. modeled the problem as a disjunctive graph and solved it using a TS algorithm [4]. Wardono and Fathi also used the TS algorithm for solving the multi-stage parallel machine problem with limited buffer capacities [45]. Wang et al. proposed a hybrid GA for a permutation flow shop scheduling problem with limited buffers [42]. Grabowski and Pempera introduced several problem-specific neighborhood structures for the problem [10]. Liu et al. developed a hybrid PSO for the same problem [25]. Furthermore, Pan et al. investigated a hybrid discrete differential evolution (DE) algorithm [30]. Lin and Ying designed a hybrid algorithm based on the features of artificial immune systems and the annealing process of simulated annealing algorithms [24].

The HFS with limited buffers (hereafter denoted the HFS-LB) is a typical scheduling problem with a strong industrial background that can be found to exist in many different industries, particularly the steel industry. However, fewer studies have been dedicated to most realistic HFS-LB problems than to the flow shop problem with limited buffers. Sawik presented a mixed-integer programming approach for makespan minimization in flexible flow lines with limited buffers [37]. Tang and Xuan designed a Lagrangian relaxation algorithm for real-time hybrid flow shop scheduling with finite intermediate buffers [40]. Wang and Tang solved the HFS with finite intermediate buffers by a hybrid algorithm combining the TS and scatter search algorithms [44]. It should be noted that the three studies discussed above considered HFS-LB problems with identical parallel machines. In a realistic industrial system, such as that associated with steelmaking casting production, there are unrelated machines with different processing abilities at each stage. Yaurima et al. solved the HFS with unrelated machines, sequence-dependent setup times, availability constraints, and limited buffers with an improved GA [47]. An HFS-LB with at most six stages was considered in [47]. However, with respect to real-world applications, more research should be conducted on large-scale HFS-LB scheduling problems. Therefore, in this study, we develop a novel hybrid algorithm combining the ABC and TS (TABC) algorithms to solve the large-scale HFS-LB scheduling problem with unrelated machines.

To the best of our knowledge, there are no previous studies that have applied the ABC algorithm to solve the HFS with limited buffers. The main features of the proposed TABC algorithm are as follows: (1) four neighborhood structures are embedded; (2) a TS-based self-adaptive neighborhood structure strategy is employed to balance the exploitation and exploration capabilities; (3) the TS-based local search heuristic is used by both employed bees and onlookers; and (4) a decoding strategy considering the limited buffer constraints is developed. The rest of this paper is organized as follows: Section 2 briefly describes the formulation of the problem. Then, the related algorithms are presented in Section 3, and Section 4 presents the two TS-based heuristics. The proposed TABC algorithm is discussed in detail in Section 5, and Section 6 presents the experimental results as well as a comparison between the proposed TABC algorithm and the best-performing algorithms reported in the literature to demonstrate the superiority of the former. Finally, Section 7 provides concluding remarks and discusses future research directions.

2. Problem descriptions

In an HFS with limited buffers, there are n jobs, m machines, s stages, and h buffer capacities between consecutive stages. Let $S = \{S_i\}_{1\leqslant i\leqslant s}$ be the series of stages, $M = \{M_k\}_{1\leqslant k\leqslant m}$ be the set of machines, $J = \{J_j\}_{1\leqslant j\leqslant n}$ be the set of jobs, and $B = \{B_q\}_{1\leqslant q\leqslant h}$ be the set of limited buffers. Let $p_{i,j,k}$ be the processing time of job j on machine k at stage i ($p_{i,j,k} \geqslant 0$), $s_{i,j}$ be the starting time of job j in stage i, and $c_{i,j}$ be the completion time of job j at stage i.

The main assumptions and constraints for the considered problem are as follows:

- In stage i, there are m_i unrelated parallel machines with different processing abilities, where $m_i \geqslant 1$.
- Between the stages i and $i+1$, there are b_i buffer capacities, where $b_i \geqslant 0$.

- Each job consists of a sequence of operations O_{ij}, where O_{ij} denotes the ith operation of job j, which should be carried out on a selected machine in stage i.
- Each job visits each stage according to the same production flow.
- When a job arrives at a stage i, it can select exactly one machine from m_i available unrelated parallel machines.
- After a job is completed at stage i, it may be processed as follows: (1) the job will be immediately delivered to the subsequent stage when one of the machines at stage $i+1$ is available; (2) in cases in which there is no available machine at stage $i+1$, the job will be stored in the following buffer if there is a buffer space; and (3) in cases in which there is not enough buffer space to store the job, the job must remain on the machine at stage i until a buffer space or a machine in the subsequent stage becomes available.
- Each machine in the same stage can process only one job at a time, and each job can be executed by only one machine at a time.
- All jobs and machines are available at time zero.
- Preemption is not allowed; that is, a job cannot be interrupted before the completion of its current operation.
- Setup times are negligible, and the problem data are deterministic and known in advance.

The aim of the paper is to schedule each job on each machine to minimize the makespan or the maximum completion time; that is, $\min c_{\max} = \max_{1\le j\le n}\{c_{s,j}\}$.

3. The related algorithms

3.1. The canonical ABC algorithm

The artificial bee colony (ABC) algorithm was proposed by Karaboga [15] to optimize multivariable and multimodal continuous functions. In the canonical ABC algorithm, three types of artificial bees are sent to given food sources to perform exploitation and exploration tasks [15–18]. Since 2005, the ABC algorithm has been applied to solve many optimization problems. Comparisons of the experimental results yielded by the ABC algorithm with those obtained using other algorithms have verified the efficiency of the ABC algorithm. Karaboga and Basturk applied the ABC algorithm to solve numerical function optimization problems [16]. Pan et al. developed a discrete version of the ABC algorithm for solving the lot-streaming flow shop scheduling problem [29]. Li et al. investigated multi-objective flexible job shop scheduling problems by using a Pareto-based ABC algorithm [21]. Mohammadi and Abadeh proposed a feature selection technique based on ABC for image steganalysis problems [26]. Tsai presented a hybrid algorithm combining ABC and bee algorithms (BAs) for constrained optimization problems [41]. Lei et al. proposed a novel artificial bee colony (ABC) clustering model in protein–protein interaction networks based on a propagating mechanism [19]. Fatih et al. applied a discrete ABC algorithm for solving no-idle permutation flow shop scheduling problem with a total tardiness criterion [7].

However, the canonical ABC algorithm, originally designed to address the continuous nature of optimization problems, cannot be used for discrete or combinatorial cases. Few studies have solved hybrid flow shop scheduling problems on a large scale by using the ABC algorithm. Furthermore, the realistic application of the HFS with limited buffers should be given more consideration, especially through the use of the efficient ABC algorithm. Therefore, in this work, certain modifications to the above-described ABC algorithm have been made to solve large-scale hybrid flow shop scheduling problems with limited buffers, as discussed in the following sections.

3.2. Tabu search algorithm

The tabu search (TS) algorithm, proposed by Glover [9], has been successfully employed to solve a large number of combinatorial optimization problems [1,3,4,8,12,20,43,45,47]. Recently, Li et al. developed a hybrid TS algorithm for multi-objective flexible job shop scheduling problems (FJSPs) [20]. Bożejko et al. proposed a parallel TS algorithm for solving HFS problems [3]. Ahonen et al. presented a hybrid algorithm combining the TS and SA algorithms for the corridor allocation problem [1]. Gao et al. applied the TS algorithm for solving distributed permutation flow shop scheduling problems [8]. Wang and Liu developed the multi-objective TS algorithm for solving the two-stage hybrid flow shop scheduling problem with preventive maintenance [43].

The TS algorithm requires an initial solution and a neighborhood structure. The algorithm executes an iterative procedure to find feasible solutions. To prevent cycling, a structure called a tabu list is introduced to prevent returning to a solution visited in the last l iterations. A detailed description of the TS algorithm can be found in [9].

4. TS-based heuristics

This section proposes two TS-based heuristics for the algorithm developed in this study. The TS-based self-adaptive neighborhood strategy is used to balance selection among the four neighborhood structures and thus to balance the exploration and exploitation abilities. The TS-based local search approach is utilized to further enhance the exploitation ability.

4.1. TS-based self-adaptive neighborhood strategy

For the permutation-based representation, neighborhood structures such as insertion, swap and pairwise exchange are commonly reported in the literature [31,38]. For the regular flow shop problem, the insertion neighborhood has been verified to be more efficient than the pairwise exchange neighborhood [31,39]. For the HFS problem, the swap neighborhood has been verified to be superior to the insertion and pairwise exchange neighborhoods. Moreover, the multi-swap neighborhood has been verified to be efficient for solving real-world HFS problems [31], mainly because with a randomly selected number of swaps, the multi-swap neighborhood can create a balance between the exploration and exploitation of the search. In addition, the inversion neighborhood is also a widely used neighborhood for permutation-based representations [38,39]. By using the inversion neighborhood, a very different individual can be generated and therefore enhance the exploration ability of the algorithm.

Because different neighborhoods play different roles in the search process, to balance the exploration and exploitation, we embed the four neighborhood structures in the proposed algorithm, i.e., swap, insertion, multi-swap, and inversion, which are hereafter denoted N_1, N_2, N_3, and N_4, respectively. Moreover, we propose a TS-based self-adaptive neighborhood strategy to balance selection among the four neighborhood structures. The detailed steps of the TS-based self-adaptive neighborhood strategy are as follows:

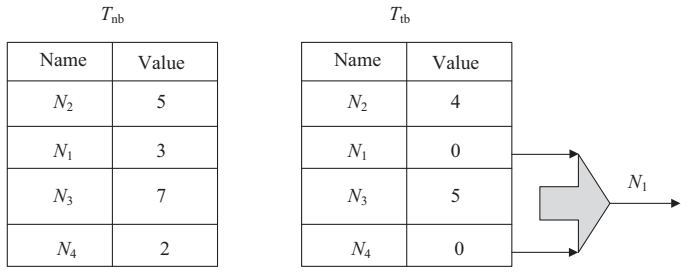
Step 1. The initialization phase: Initialize the neighborhood structure table (T_{nb}) with the four abovementioned neighborhood structures in a random order, with each element v_{nb} set to zero. Initialize the neighborhood tabu table (T_{tb}) with each element v_{tb} set to zero. Initialize the parameter v_{max}, which is the maximum value for T_{tb}.
Step 2. The neighborhood structure selection mechanism: From T_{nb}, select the neighborhood structure that has not been tabued ($v_{tb} = 0$), with the maximum v_{nb} value. If there is more than one neighborhood structure with the same maximum v_{nb} value, randomly select one as the current neighborhood structure. Fig. 1(a) illustrates an example of the neighborhood selection procedure, in which N_1 is selected.
Step 3. Update the tabu table: Decrease each v_{tb} value (larger than zero) by one for each neighborhood structure in the current T_{tb} table.
Step 4. Neighboring solution generation: Using the selected neighborhood structure, generate several neighboring solutions, and evaluate the newly generated solutions. If the new solution is better than the current solution, let $v_{nb} = v_{nb} + 1$ for the selected neighborhood structure. Fig. 1(b) illustrates an example for this situation. Otherwise, set the corresponding value v_{tb} to v_{max} (e.g., $v_{max} = 10$), which means that the neighborhood structure that cannot find better solutions should be tabued over several iterations. Fig. 1(c) provides an example for this situation, in which the initial statuses of T_{nb} and T_{tb} are set to the same as those shown in Fig. 1(a).

By using the above-described self-adaptive neighborhood strategy, the neighborhood structure that is solely used to generate a better solution will be set to a higher v_{nb} value and will then have a higher probability of being selected in the next run; otherwise, the neighborhood structure with a worse result will be set to a higher v_{tb} value to tabu it over several iterations. Therefore, the proposed algorithm will have a better exploitation capability. If the algorithm falls into a local optimum, i.e., no neighborhood structure can obtain a better solution, then all of the neighborhood structures will be tabued. At that time, to enhance the exploration ability, we will randomly select a neighborhood structure to generate neighboring solutions. Fig. 1(d) presents an example for this situation, in which the algorithm falls into a local optimum and each neighborhood structure has the same possibility of being selected. In Fig. 1(d), N_2 is used first, followed by the N_3 neighborhood. At the same time, the v_{tb} value for the other neighborhood structure will be updated. The above-described process loops until a better solution is found or the stop condition is satisfied. Therefore, the above-discussed self-adaptive strategy can balance the exploitation and exploration capabilities.

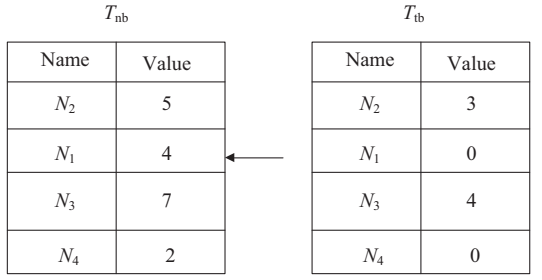
4.2. TS-based local search approach

In the proposed algorithm, the TS-based local search is used to conduct the search task for both employed bees and onlookers. Given a current solution, the main steps of the TS-based local search, denoted as *TS_LocalSearch*(s), are as follows.

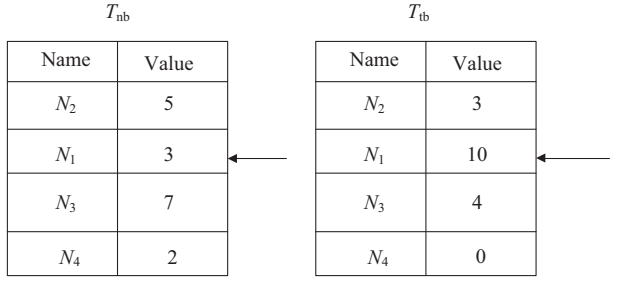
Step 1. Set the system parameters.
Step 2. Set $j = 0$; perform steps 3–6 until $j \geqslant T_{max}$, where T_{max} is the maximum number of iterations for which the best solution found thus far has not been improved.
Step 3. For $i = 1$ to *numNS*, perform the sub-steps from 3.1 to 3.2.
Step 3.1. Select a neighborhood structure by using the TS-based self-adaptive neighborhood strategy explained previously.
Step 3.2. Produce a neighboring solution by using the selected neighborhood structure, and insert it into the queue denoted *qNS*.
Step 4. Sort the solutions in *qNS* in increasing order according to their makespan value.
Step 5. Select the best neighboring solution as the current solution satisfying one of the following conditions:
(1) the first solution in *qNS* that is non-tabued.
(2) if this does not exist, select the first solution in *qNS*.



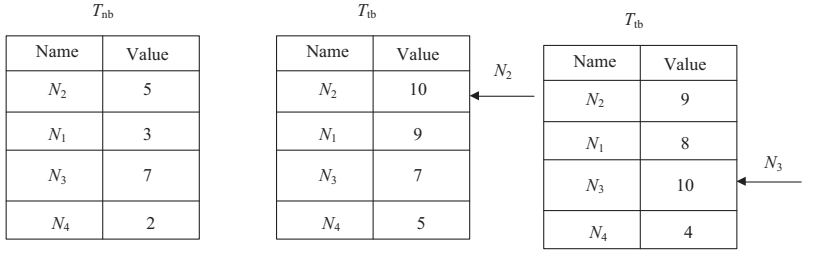
(a) Selection of the neighborhood structure (N_1 is selected)



(b) Adjustment after generating a better neighboring solution by using N_1



(c) Adjustment after generating a worse neighboring solution by using N_1



(d) Adjustment after falling into a local optimum

Fig. 1. Self-adaptive strategy.

Step 6. Update the tabu list by adding the selected neighboring solution and remove the oldest solution if the tabu list has overflowed or the duration of the oldest solution in the tabu list exceeds the termination criterion T_{max}.

5. The proposed TABC

In this section, we present the pseudo-code of the TABC algorithm in Fig. 2. Then, we detail the components of the TABC algorithm in the following subsections, which include the encoding and decoding approach, the population initialization method, and the strategies for the three types of artificial bees.

5.1. Encoding

Similarly to the approach described in Refs. [31,35], to solve the HFS with limited buffers, 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 job

begin

Initialize the system parameters.

Initialize the population, and evaluate each solution in the population.

Select the best solution $\pi(b)$ and the worst solution $\pi(w)$.

while maximum computational time is not reached **do**

1) Apply the employed bee strategy (cf. Section 5.4): Randomly select a solution $\pi(i)$. Obtain $\pi(i')$ by performing $TL\_LocalSearch(\pi(i))$. Evaluate $\pi(i')$ and use it to update $\pi(b)$ and $\pi(i)$. Update T_{nb} and T_{tb}.

2) Apply the onlooker bee strategy (cf. Section 5.5): Randomly select two solutions $\pi(x)$ and $\pi(y)$, and select the best one as $\pi(j)$. Obtain $\pi(j')$ by performing $TL\_LocalSearch(\pi(j))$. Evaluate $\pi(j')$, and use it to update $\pi(b)$, $\pi(w)$ and $\pi(j)$. Update T_{nb} and T_{tb}.

3) Apply the scout bee strategy (cf. Section 5.6): Select the solution $\pi(i)$ that has not been improved during the last l_{max} trials. Obtain $\pi(b')$ by performing a random local search N_s times around $\pi(b)$. Evaluate $\pi(b')$, and use it to update $\pi(b)$ and $\pi(i)$.

end

end

Fig. 2. Pseudo-code of the TABC algorithm.

number. Given an HFS-LB problem in a real steel-casting production system, there are five jobs, three stages, and a total of seven machines. Table 1 presents the processing times and the limited buffer capacities. Suppose one solution is represented by {1,5,3,2,4}, which means that in the first stage, the scheduling sequence is J_1, J_5, J_3, J_2, and J_4. To schedule each operation, the first available machine is selected. In the following stages, the operations will be scheduled in a first-come–first-serve manner. The machine assignment rule is the same as that of the first stage, i.e., select the first available machine for each operation.

5.2. Decoding with limited buffer

It is clear that the solution representation outlined above contains no information about the machine assignment. In the proposed algorithm, as described in Refs. [40,44,47], each job, when it arrives at each stage, will be assigned to the first available machine in the current stage. The detailed decoding steps are as follows.

Step 1. For each job j in the current solution representation, perform the following steps.
Step 2. For each stage i, except for the last stage, select the machine k_i^* at stage i with the earliest available time for job j; let $s_{i,j}$ and $c_{i,j}$ be the possible starting time and completion time for job j at stage i without considering the limited buffer between the stages i and $i+1$. Let $u_{i,i+1}$ be the earliest available time of the limited buffer between the stages i and $i+1$; select the machine k_{i+1}^* at stage $i+1$ with the earliest available time for job j; let $s_{i+1,j}$ be the possible starting time for job j at stage $i+1$ without considering the limited buffer between the stages i and $i+1$.
Step 3. Compute the starting and completion times of job j at consecutive stages i and $i+1$ according to one of the two conditions as follows.
Step 3.1. If $s_{i+1,j} \leqslant c_{i,j}$, that is, job j can be transferred directly to the next stage without any waiting time, then memorize the two groups of starting and completion times $[s_{i,j}, c_{i,j}]$ and $[c_{i,j}, c_{i,j} + p_{i+1,j,k_{i+1}^*}]$, where p_{i+1,j,k_{i+1}^*} is the processing time of job j on machine k_{i+1}^* at stage $i+1$.

Table 1
Processing time table.

Stage	Buffer	Machine	J_1	J_2	J_3	J_4	J_5
Stage 1		M_1	30	31	30	32	40
		M_2	28	32	31	33	39
	2						
Stage 2		M_3	25	26	29	31	45
		M_4	27	32	35	31	38
		M_5	25	28	29	31	39
	2						
Stage 3		M_6	31	32	27	25	40
		M_7	30	30	30	35	38

Step 3.2. If $s_{i+1,j} > c_{i,j}$, then one of the following three conditions should be considered:

(1) If $u_{i,i+1} \leqslant c_{i,j}$, that is, there exists an available buffer for the completed operation, then memorize $[s_{i,j}, c_{i,j}]$ and let it be the first block that is processed on machine k_i^*; let $[c_{i,j}, s_{i+1,j}]$ be the second block that remains on the available buffer, and let $[s_{i+1,j}, c_{i+1,j}]$ be the third block on machine k_{i+1}^*. Fig. 3(a) gives a Gantt chart under this condition. In Fig. 3(a), the completion time $c_{1,2}$ is greater than the buffer available time $u_{1,2}$, whereas $c_{1,2}$ is less than the starting time $s_{2,2}$ in the next stage. Therefore, for job J_2, we can assign an immediate buffer between the two stages. The yellow filled rectangle indicates that the corresponding operation should remain on the buffer.

(2) If $c_{i,j} < u_{i,i+1} < s_{i+1,j}$, there exists an available buffer a few times after the completion of the operation. Therefore, we should memorize four blocks, i.e., $[s_{i,j}, c_{i,j}]$ and $[c_{i,j}, u_{i,i+1}]$ on machine k_i^*, $[u_{i,i+1}, s_{i+1,j}]$ on the available buffer, and $[s_{i+1,j}, c_{i+1,j}]$ on machine k_{i+1}^*. Fig. 3(b) presents the situation illustrating this condition. In Fig. 3(b), there is no available buffer when J_4 completes its work in the first stage. After J_3 releases the buffer, J_4 can remain in the buffer before its starting time in the next stage. Therefore, the processing of J_4 at the two stages should be divided into four blocks.

(3) If $u_{i,i+1} \geqslant s_{i+1,j}$, no available buffer exists, and we should memorize three blocks, i.e., $[s_{i,j}, c_{i,j}]$, and $[c_{i,j}, s_{i+1,j}]$ on machine k_i^* and $[s_{i+1,j}, c_{i+1,j}]$ on machine k_{i+1}^*. Fig. 3(c) displays the Gantt chart under this condition. In Fig. 3(c), J_2 is in the buffer after J_4 completes its work in the first stage. The available time of the buffer is equal to the starting time of J_4 in the next stage. Therefore, J_4 has no possibility of remaining in the buffer and should remain in the first stage until the starting time of the second stage.

Given the solution {1,5,3,2,4} to the example problem discussed above, Fig. 4 exhibits the corresponding Gantt chart for the solution. In the Gantt chart, each operation is marked by a rectangle with the job number, and each machine is labeled with the machine number. There are two limited buffers for each pair of consecutive stages. Each stage is divided by lines that represent the limited buffer between two consecutive stages. Each number listed below the rectangle indicates the completion time of the corresponding operation. For example, in the Gantt chart, there are three stages, where 'M1' and 'M2' are the two parallel machines in the first stage and 'M6' and 'M7' are grouped in the last stage. In the first stage, J_1, J_5, J_3, J_2, and J_4 are scheduled one by one according to their sequence in the solution. In the second stage, each operation can be processed immediately after its completion of the first stage; therefore, there is no operation that should remain on the limited buffer between the two consecutive stages. However, in the last stage, job J_2 must wait after its completion of the second stage. Therefore, J_2 remains on the limited buffer between the second stage and the last stage.

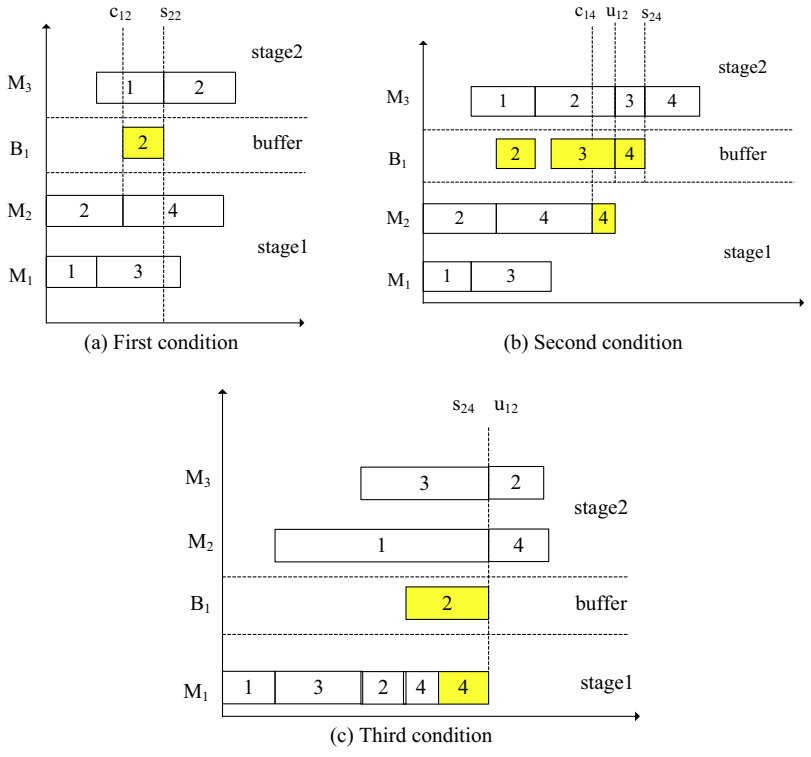


Fig. 3. Gantt chart for the three situations.

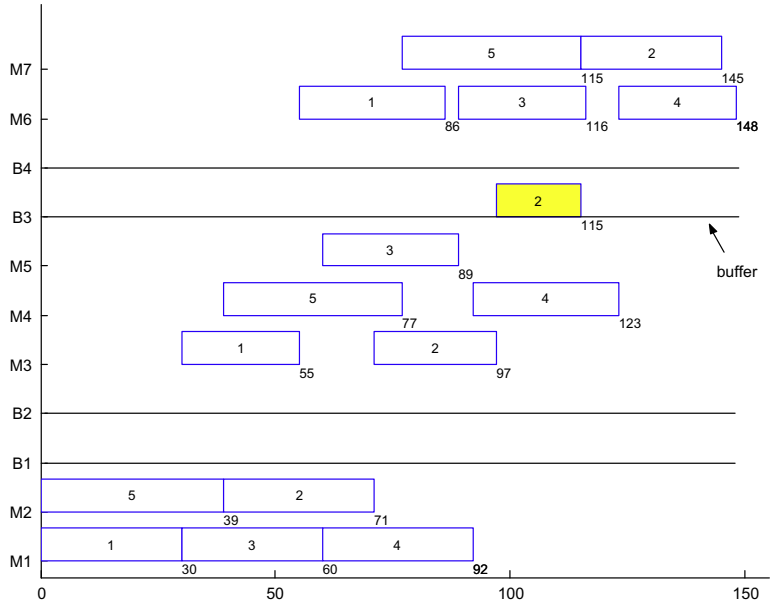


Fig. 4. Gantt chart for the example problem ($f = 148$).

5.3. Population initialization

The TABC algorithm begins with a population of P_s initial solutions. A population with a high level of solution quality and diversity is crucial for the algorithm, and we initialize the population as follows:

Step 1. Let counter $Cnt = 1$; perform the following steps until $Cnt = P_s$.
Step 2. Generate a solution in a random manner 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 3. Go back to step 2.

5.4. Employed bee strategy

Employed bees complete the exploitation task around their given solutions. In the proposed algorithm, each employed bee is set to a randomly selected solution to complete the search function. The detailed steps executed by each employed bee are as follows.

Step 1. Randomly select a food source in the current population.
Step 2. Implement the TS-based local search approach, as discussed in Section 4.2, around the given food source.
Step 3. Evaluate each newly generated food source, and perform the following replacement processes: (1) if its fitness value is better than the best solution found thus far, replace the latter; (2) if its fitness value is better than the current food source, replace the latter.

5.5. Onlooker bee strategy

Onlooker bees aim to perform further search tasks after collecting the results of employed bees. Unlike the employed bee strategy, the onlooker strategy aims to find a better food source that can replace the current individual or the worst individual in the current population. Therefore, the onlooker bee strategy causes the whole population to exploit more promising regions and therefore can enhance the exploitation ability of the whole population. In this study, we propose the following novel method for each onlooker bee:

Step 1. Randomly select two food sources in the current population, and select the better one for the current onlooker bee.
Step 2. Implement the TS-based local search approach around the current individual.
Step 3. Evaluate each newly generated food source, and perform the following replacement processes: (1) if its fitness value is better than the current individual, replace the latter; (2) if its fitness value is better than the worst solution in

the current population, replace the latter; and (3) if its fitness value is better than the best solution found thus far, replace the latter.

5.6. Scout bee strategy

In the canonical ABC algorithm, a solution without improvement after a certain number of generations becomes an abandoned one, and a scout bee is subsequently produced for further exploitation. In this study, we apply the following steps for each scout bee:

Step 1. For the best solution found thus far, perform the exploitation process N_s times by using a randomly selected neighborhood structure.
Step 2. Evaluate the newly generated food source, and perform the following replacement processes: (1) if its fitness value is better than the best solution found thus far, replace the latter; (2) replace the source food directly.

6. Numerical results

This section discusses the computational experiments employed to evaluate the performance of the proposed algorithm. Our algorithm is implemented in C++ on an Intel Core i7 3.4 GHz PC with 16 GB of memory. The compared algorithms include the GAH (Ruiz and Maroto [34]), the GAS (Yaurima et al. [47]), the TSSCS (Wang and Tang [44]), the TS (Wardono and Fathi [45]), and the DABC (Pan et al. [29]) algorithms. It should be noted that the TS, GAS, and TSSCS algorithms are three efficient algorithms for the HFS with limited buffer constraints, whereas the GAH and DABC algorithms are promising algorithms for scheduling problems with realistic constraints. To the best of our knowledge, there is no algorithm that considers the HFS with limited buffer under unrelated parallel machines; therefore, we code the above five algorithms and adopt the parameter settings proposed in the corresponding references, with the exception that the computational times for each instance are set to 150 s. For each compared algorithm, the same initialization method and decoding strategy for limited buffers are utilized. Each compared algorithm is independently run for five replications (30 runs for each replication) with each instance. The performance measure is a 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 any algorithm and C_c is the best solution collected by a given compared algorithm.

6.1. Experimental instances

In this study, we generate 36 problem instances with different scales according to practical industrial situations. The detailed instance parameters are as follows:

- To make the problem reflect realistic production more closely, we divide the stages into six types, i.e., 5, 10, 15, 20, 25, and 30, in the shop.
- There are 5–10 unrelated parallel machines in each stage; that is, the parallel machines in each stage have different processing abilities.
- There are 20, 40, 60, 80, 100, and 200 jobs in the shop.
- For each type of stage and each type of job, we independently generate one instance with a randomly selected number of parallel machines. Therefore, there are 36 problem instances with different problem scales, which are denoted from "Case0" to "Case35".

6.2. Experimental parameters

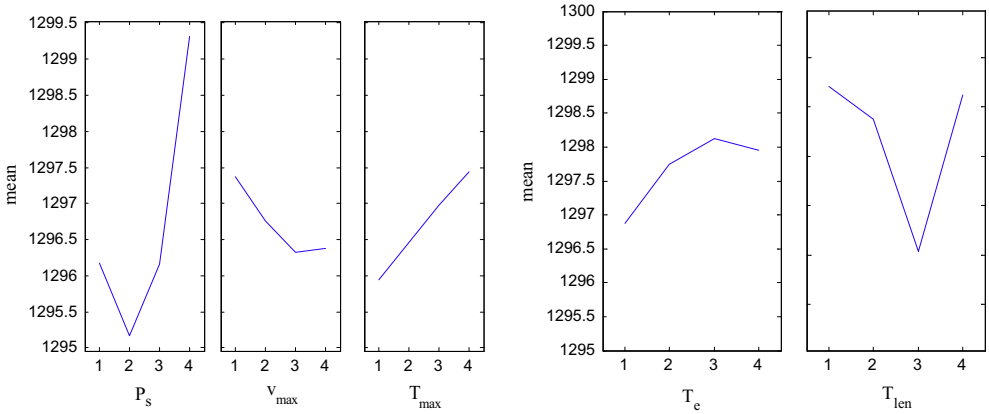
In this study, the proposed algorithm has several parameters. Preliminary experiments and analyses indicate that five key parameters play an important role in determining the performance of the algorithm. According to Refs. [29,30] and our preliminary experiments, the other less important parameters are set as follows: (1) the numbers of employed bees, onlooker

Table 2
Combinations of the key parameter values.

Parameter	Level			
	1	2	3	4
P_s	5	10	20	50
v_{max}	1	2	5	10
T_{max}	3	5	10	20
T_e	$n \times m/2$	$n \times m/4$	$n \times m/8$	$n \times m/10$
T_{len}	$n \times m/2$	$n \times m/4$	$n \times m/8$	$n \times m/10$

bees, and scout bees are set to P_s, P_s, and 1, respectively; (2) the limited number of cycles through which a food source cannot be further improved and thereby a scout bee generated is set to $l_{max} = 20$; (3) the number of search processes performed by a scout bee is set to $N_s = 10$; (4) the number of neighboring solutions to be generated is set to $numNS = P_s$; and (4) the stop condition is set to point at which the maximum computational time exceeds 150 s.

To set the five key parameters, the Taguchi method of DOE [27] is utilized to test effects of these parameters on the performance of the proposed algorithm. The levels of the five parameters are presented in Table 2. In Table 2, n represents the



(a) Factor-level trend for P_s v_{max} and T_{max}

(b) Factor-level trend for T_e and T_{len}

Fig. 5. Factor-level trend for the five key parameters.

Table 3
Comparison of the RPI values for $TABC_{NA}$ and TABC.

Instance	Scale	$TABC_{NA}$	TABC
Case0	20.vs.5	2.14	**1.79**
Case1	20.vs.10	1.25	**0.00**
Case2	20.vs.15	0.59	**0.15**
Case3	20.vs.20	1.36	**0.68**
Case4	20.vs.25	1.10	**0.37**
Case5	20.vs.30	0.70	**0.31**
Case6	40.vs.5	0.76	**0.76**
Case7	40.vs.10	1.13	**0.00**
Case8	40.vs.15	1.17	**0.23**
Case9	40.vs.20	0.75	**0.00**
Case10	40.vs.25	1.10	**0.24**
Case11	40.vs.30	0.67	**0.20**
Case12	60.vs.5	1.13	**0.23**
Case13	60.vs.10	1.09	**0.14**
Case14	60.vs.15	1.30	**0.40**
Case15	60.vs.20	0.48	**0.40**
Case16	60.vs.25	0.69	**0.00**
Case17	60.vs.30	0.37	**0.06**
Case18	80.vs.5	1.17	**0.00**
Case19	80.vs.10	0.94	**0.10**
Case20	80.vs.15	1.02	**0.00**
Case21	80.vs.20	0.70	**0.21**
Case22	80.vs.25	0.64	**0.58**
Case23	80.vs.30	0.00	0.11
Case24	100.vs.5	0.89	**0.45**
Case25	100.vs.10	0.79	**0.26**
Case26	100.vs.15	0.98	**0.08**
Case27	100.vs.20	0.83	**0.32**
Case28	100.vs.25	0.75	**0.63**
Case29	100.vs.30	0.35	**0.20**
Case30	200.vs.5	0.75	**0.35**
Case31	200.vs.10	0.26	**0.26**
Case32	200.vs.15	0.60	**0.00**
Case33	200.vs.20	0.25	**0.00**
Case34	200.vs.25	1.05	**0.00**
Case35	200.vs.30	0.80	**0.00**
Mean		0.85	**0.26**

total number of jobs, and m represents the total number of machines in the system. Because the five parameters are set with four factor levels, two independent tests are performed to determine the parameter settings. The first test is performed to determine the first three parameters, for which an orthogonal array $L_{16}(4^3)$ is selected. The second test utilized an orthogonal array $L_{16}(4^2)$ to set the following two parameters. Each test is performed 30 times independently; then, the average makespan value obtained by the proposed algorithm is collected as the response variable (RV). Fig. 5(a) presents the factor-level trend for the three key parameters, i.e., the population size P_s, the maximum value of the neighborhood tabu table element

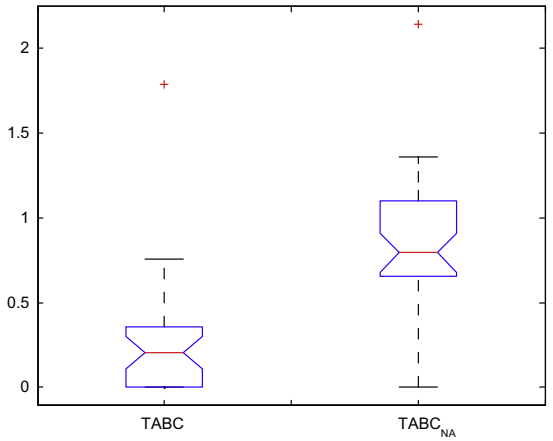


Fig. 6. Means and 95% LSD intervals for TABC and $TABC_{NA}$ ($p = 2.5 \times 10^{-9}$).

Table 4
Comparison of the RPI values for the six algorithms.

Instance	Scale	GAH	GAS	DABC	TS	TSSCS	TABC
Case0	20.vs.5	6.07	2.50	**0.00**	3.57	1.07	1.79
Case1	20.vs.10	4.18	1.25	1.04	1.67	0.42	**0.00**
Case2	20.vs.15	1.91	1.47	0.29	1.03	**0.00**	0.15
Case3	20.vs.20	2.72	0.79	0.45	0.79	**0.34**	0.68
Case4	20.vs.25	2.01	1.19	0.37	1.10	**0.00**	0.37
Case5	20.vs.30	1.63	0.85	**0.00**	0.93	0.39	0.31
Case6	40.vs.5	3.55	1.78	**0.51**	1.27	1.78	0.76
Case7	40.vs.10	1.78	0.65	0.81	1.78	1.29	**0.00**
Case8	40.vs.15	2.46	1.06	**0.00**	0.47	0.12	0.23
Case9	40.vs.20	1.50	0.37	1.50	0.94	0.75	**0.00**
Case10	40.vs.25	1.49	1.49	0.39	1.42	1.10	**0.24**
Case11	40.vs.30	0.87	**0.20**	0.54	0.80	0.40	**0.20**
Case12	60.vs.5	3.17	1.36	**0.23**	1.81	1.36	**0.23**
Case13	60.vs.10	3.00	1.36	0.68	1.91	1.09	**0.14**
Case14	60.vs.15	3.09	1.20	1.20	1.80	1.70	**0.40**
Case15	60.vs.20	1.69	1.05	0.73	0.81	0.48	**0.40**
Case16	60.vs.25	1.53	0.97	0.97	0.97	0.90	**0.00**
Case17	60.vs.30	1.43	0.68	0.62	0.62	0.68	**0.06**
Case18	80.vs.5	2.83	1.17	0.67	0.83	0.83	**0.00**
Case19	80.vs.10	1.56	0.94	0.94	0.83	1.14	**0.10**
Case20	80.vs.15	1.58	1.58	1.11	0.19	1.11	**0.00**
Case21	80.vs.20	2.25	0.63	0.77	0.56	1.19	**0.21**
Case22	80.vs.25	1.80	0.39	0.39	0.71	**0.00**	0.58
Case23	80.vs.30	1.08	0.27	0.38	0.54	0.38	**0.11**
Case24	100.vs.5	2.79	0.67	1.00	1.11	**0.45**	**0.45**
Case25	100.vs.10	1.93	1.49	1.58	1.23	1.23	**0.26**
Case26	100.vs.15	1.58	0.60	**0.00**	0.90	0.60	0.08
Case27	100.vs.20	1.41	0.64	0.90	0.45	0.96	**0.32**
Case28	100.vs.25	1.03	0.98	1.21	1.32	1.44	**0.63**
Case29	100.vs.30	0.55	**0.00**	0.05	0.25	0.10	0.20
Case30	200.vs.5	1.16	0.52	0.87	0.81	0.40	**0.35**
Case31	200.vs.10	1.16	0.47	0.74	0.47	0.58	**0.26**
Case32	200.vs.15	0.51	0.42	0.23	0.19	0.37	**0.00**
Case33	200.vs.20	0.64	0.42	0.51	0.47	0.21	**0.00**
Case34	200.vs.25	0.66	0.39	0.35	0.35	0.08	**0.00**
Case35	200.vs.30	2.75	2.17	**0.00**	0.23	0.50	**0.00**
Mean		1.98	0.94	0.61	0.98	0.71	**0.26**

v_{max}, and the maximum generations without improvement to stop the local search T_{max}. Fig. 5(b) reports the factor-level trend for the two parameters, i.e., the tabu tenure T_e and the tabu list size T_{len}. From the factor-level trend, we can conclude that the proposed algorithm will perform better under the following values of the five key parameters: $P_s = 10$, $v_{max} = 5$, $T_{max} = 3$, $T_e = n \times m/2$, and $T_{len} = n \times m/8$.

6.3. Effectiveness of the proposed self-adaptive strategy

To verify the effectiveness of the proposed TS-based self-adaptive strategy, in this section, we conduct a detailed comparison of the two algorithms, i.e., the TABC algorithm with a TS-based self-adaptive strategy, hereafter denoted TABC, and the TABC algorithm without the TS-based self-adaptive strategy, hereafter denoted $TABC_{NA}$. It should be noted that $TABC_{NA}$ randomly chooses one of the neighborhood structures discussed in Section 4.1. The parameters of the two algorithms are the same as those discussed in Section 6.2. To better estimate the performance of the two compared algorithms, a total of five replications for each instance are carried out. Then, the RPI results, averaged across the five replications for each instance, are reported in Table 3.

Table 3 contains four columns. The first column presents the instance name. The following column presents the scale of the problem, where the two numbers correspond to the number of jobs and number of stages, respectively. The last two columns in the table present the average RPI values obtained by each compared algorithm. It can be observed from Table 3 that TABC improves the average RPI value of $TABC_{NA}$ for 33 out of 36 instances and leads to an overall average RPI value equal to 0.26%, which is approximately four times lower than that of $TABC_{NA}$. In addition, for solving the other two instances, i.e., "Case6" and "Case31", the proposed TABC obtains the same RPI values as those obtained by $TABC_{NA}$. However, TABC does not surpass $TABC_{NA}$ for all of the instances and produces a slightly worse RPI value for "Case23". This result might due to the instance favoring local exploitation, whereas TABC, with its enhanced strategies, stresses more exploration in the search

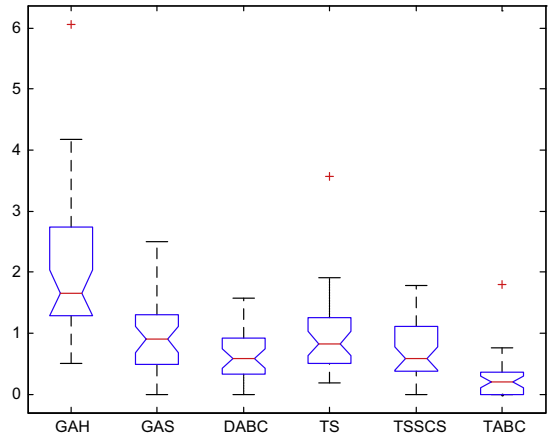


Fig. 7. Means and 95% LSD intervals for the six compared algorithms.

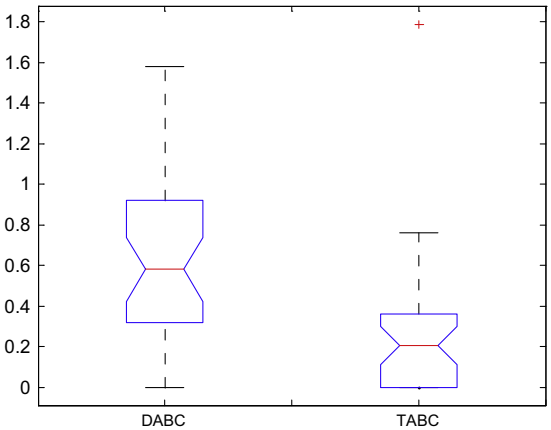


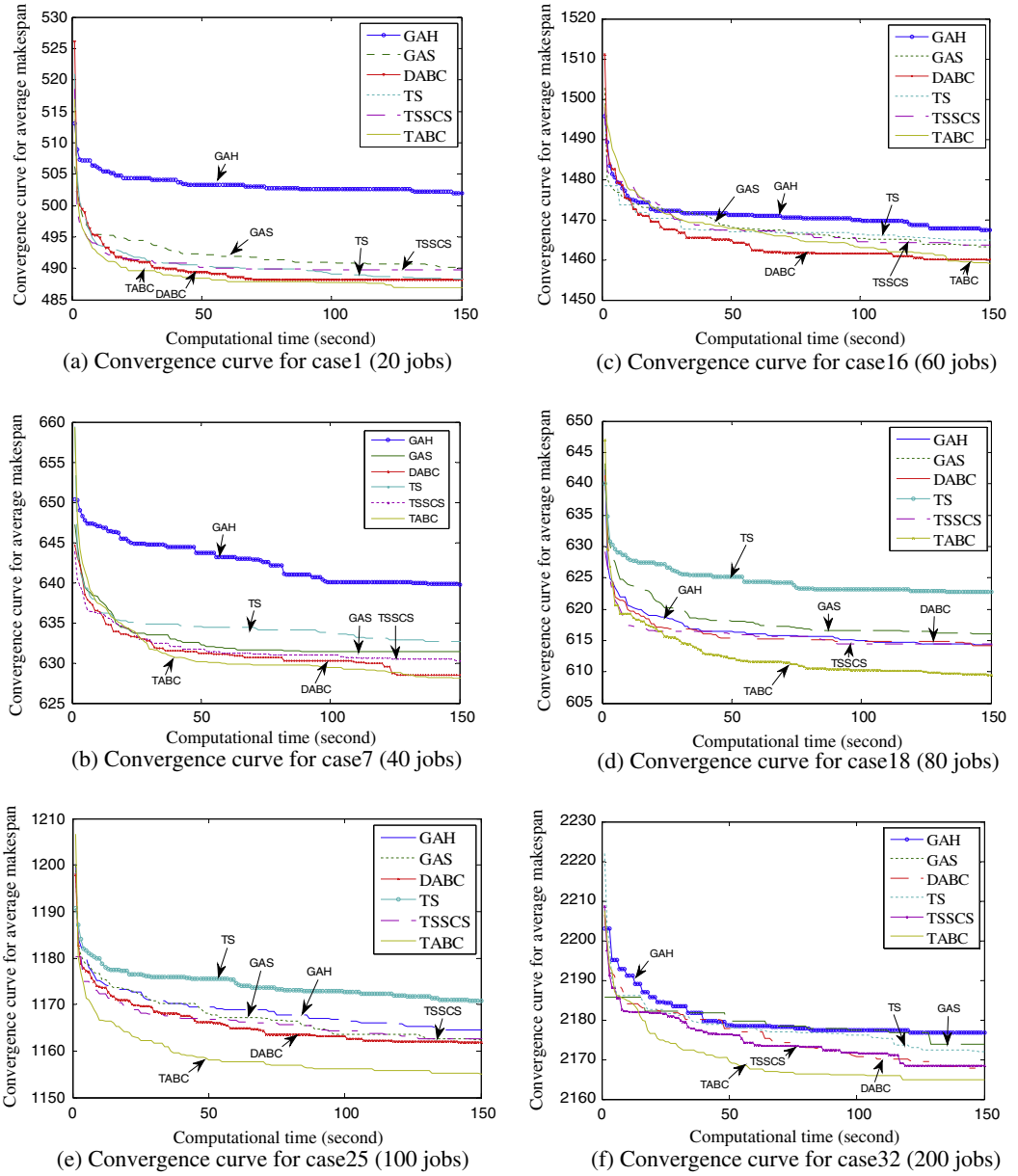
Fig. 8. Means and 95% LSD intervals for TABC and DABC ($p = 0.003$).

process. Because the 36 test instances with different scales are generated randomly, it can be concluded that, on average, the TS-based self-adaptive strategy improves the performance of the $TABC_{NA}$ algorithm by a substantial margin.

To verify whether the observed differences persist in the statistical sense, ANOVA is also conducted. Fig. 6 reports the means and 95% intervals for TABC and $TABC_{NA}$. It can be concluded from Fig. 6 that the two compared algorithms result in statistically significant differences in the response variable RPI at a 95% confidence level, with the p-value 2.5×10^{-9}. Therefore, the proposed TABC is statistically significantly better than $TABC_{NA}$, which verifies the effectiveness of the proposed TS-based self-adaptive strategy.

6.4. Comparisons with other presented algorithms

To verify the efficiency and effectiveness of the proposed TABC algorithm, in this section, we select five presented algorithms, i.e., GAH, GAS, DABC, TS, and TSSCS, to conduct a detailed comparison. Each compared algorithm is independently run five times for each instance. We compute the RPI for each replication and report the average RPI values across the five replications for each instance in Table 4 (the minimum RPI values are in bold). Moreover, the average performance for each compared algorithm is also collected to make a detailed comparison of the convergence abilities of the algorithms.



(a) Convergence curve for case1 (20 jobs)

(c) Convergence curve for case16 (60 jobs)

(b) Convergence curve for case7 (40 jobs)

(d) Convergence curve for case18 (80 jobs)

(e) Convergence curve for case25 (100 jobs)

(f) Convergence curve for case32 (200 jobs)

Fig. 9. Convergence curve for the different scale problems.

Table 4 contains eight columns. The last six columns in the table present the average RPI values obtained by each compared algorithm. It can be observed from Table 4 that (1) for solving all 36 instances, TABC obtains 26 optimal results, whereas the numbers for GAH, GAS, DABC, TS, and TSSCS are 0, 2, 7, 0, and 5, respectively. It should be noted that for solving medium- and large-scale problems, the proposed algorithm exhibits better performance. For example, for solving the six 200-job instances, TABC obtains all of the optimal values. That is, TABC is suitable for solving large-scale problems. (2) Compared with GAH, the proposed algorithm yields superior results for all instances. (3) Compared with the GAS algorithm, TABC yields better solutions for 34 out of 36 instances. (4) Compared with the recently published DABC algorithm, we can observe two distinct results: in solving the problems at small scales, such as the 20-job and 40-job instances, TABC and DABC exhibit similar performances, whereas for the medium- and large-scale instances, TABC performs notably better than DABC. (5) Compared with the TS algorithm, TABC yields 36 optimal results out of 36 instances. (6) TSSCS performs slightly better when solving the 20-job problems, whereas TABC obtains better results for the relatively large-scale problems than those obtained by the TSSCS algorithm. (7) Finally, the last row presents the average performance of the RPI value for each of the compared algorithms. The average performance indicates that, considering the search ability for the given 36 problems, TABC demonstrates the best performance among the six compared algorithms, followed by DABC, TSSCS, GHS, TS, and GAH successively.

Next, to verify whether the observed differences are statistically significant, we carry out ANOVA, in which the type of algorithm is considered as a factor. Fig. 7 presents the means and the 95% LSD intervals for the six compared algorithms. It can be observed from Fig. 7 that the proposed TABC is significantly better than the other five compared algorithms. The algorithm exhibiting the next best performance is DABC, which clearly performs better than the other four compared algorithms. To conduct a further comparison with the DABC algorithm, we also carry out ANOVA for the two algorithms. Fig. 8 reports the means and 95% LSD intervals for TABC and DABC. The resulting p-value equals 0.003, which verifies that the proposed TABC is better than DABC in terms of statistical significance. It can be concluded from the foregoing analysis that, with respect to search performance, the proposed algorithm is competitive with the other algorithms, especially when solving large-scale HFS problems.

To further verify the convergence performance of the proposed TABC, we collect the average performance of the six compared algorithms for solving problems of different scales. Six instances with problems of different scales are randomly selected to make detailed comparisons, i.e., the 20-job, 40-job, 60-job, 80-job, 100-job, and 200-job instances. Figs. 9(a)–(f) report the convergence curves for the six instances. It can be observed from the convergence curves that for solving instances of different scales, the proposed TABC exhibits a better convergence ability, especially for large-scale problems,

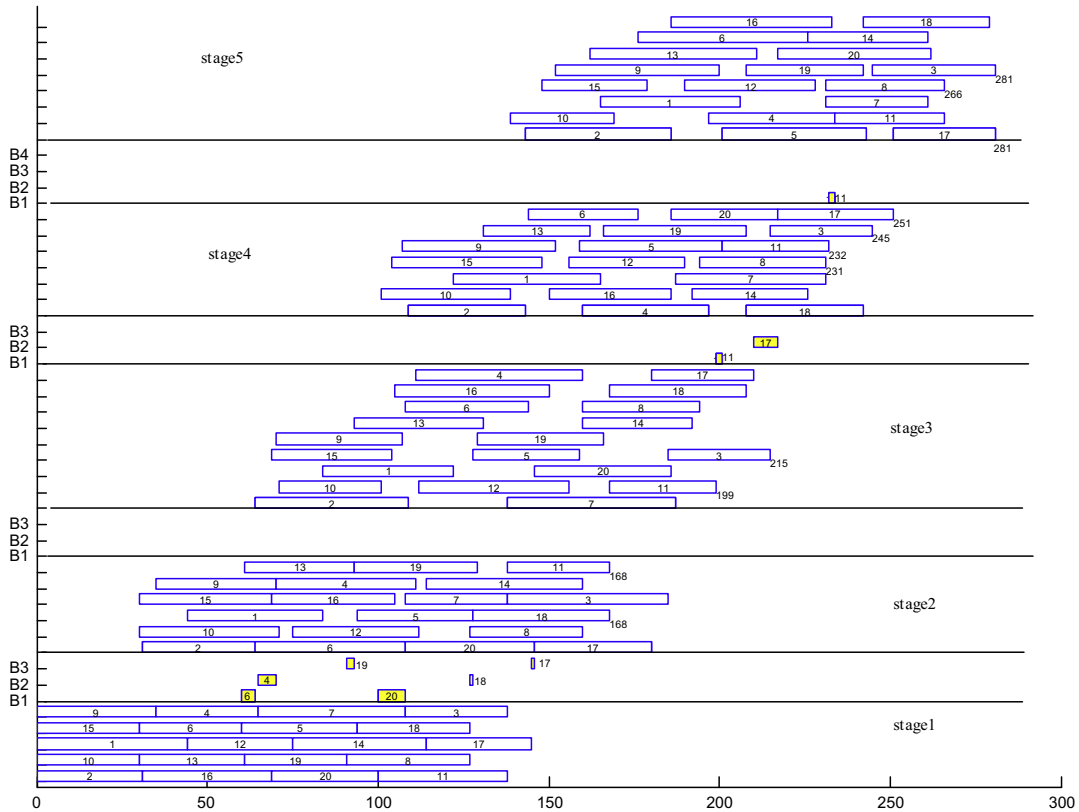


Fig. 10. Gantt chart for the best solution for Case0 (f = 281).

e.g., the 80-job, 100-job, and 200-job instances. The comparisons of the convergence abilities of the algorithms further verify the robustness and efficiency of the proposed TABC algorithm. Fig. 10 presents the Gantt chart for the best solution for "Case0" obtained by TABC.

6.5. Experimental analysis

Based on the comparisons discussed above, we can observe that the proposed TABC algorithm is suitable for solving large-scale HFS-LB problems. The main reasons are as follows: first, in the proposed algorithm, four different neighborhood structures are embedded to balance the exploitation and exploration abilities of the algorithm; second, the TS-based neighborhood structure selection strategy imparts to the proposed algorithm the ability to select a suitable neighborhood structure during the evolutionary stages, which can further balance the exploration and exploitation abilities of the algorithm; third, the TS-based local search method enhances the search ability of both employed bees and onlookers; and finally, the simple scout strategy further improves the exploration ability of the proposed algorithm while maintaining the algorithms exploitation ability and can thus enable TABC to easily escape local optima.

7. Conclusion

In this study, we propose a hybrid algorithm combining ABC and TS algorithms for solving hybrid flow shop problems with limited buffers. Based on the experimental comparisons performed, we can conclude that the proposed algorithm exhibits high exploitation and exploration capabilities in solving large-scale HFS-LB problems. The main advantages of the proposed algorithms are as follows: (1) the TS-based self-adaptive neighborhood strategy is embedded in TABC, which can balance the exploitation and exploration abilities of the algorithms; (2) the TS-based local search is applied to the employed bees and onlookers with different functions, which can further enhance the exploitation ability of the proposed algorithm. Future work will focus on how to enhance the balance between exploitation and exploration and the reliability of the simulation results, as well as how to apply the proposed algorithm to solve other potential realistic applications, such as the HFS under reactive conditions and the HFS in uncertain environments.

References

[1] H. Ahonen, A.G. Alvarenga, A.R.S. Amaral, Simulated annealing and tabu search approaches for the corridor allocation problem, Eur. J. Oper. Res. 232 (1) (2014) 221–233.
[2] A. Babayan, D. He, Solving the n-job three-stage flexible flowshop scheduling problem using an agent-based approach, Int. J. Prod. Res. 42 (4) (2004) 777–799.
[3] W. Bożejko, J. Pempera, C. Smutnicki, Parallel tabu search algorithm for the hybrid flow shop problem, Comput. Ind. Eng. 65 (3) (2013) 466–474.
[4] P. Brucker, S. Heitmann, J. Hurink, Flow-shop problems with intermediate buffers, OR. Spectrum. 25 (4) (2003) 549–574.
[5] F. Dugardin, F. Yalaoui, L. Amodeo, New multi-objective method to solve re-entrant hybrid flow shop scheduling problem, Eur. J. Oper. Res. 203 (1) (2010) 22–31.
[6] O. Engin, G. Ceran, M.K. Yilmaz, An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems, Appl. Soft. Comput. 11 (3) (2011) 3056–3065.
[7] T.M. Fatih, Q.K. Pan, P.N. Suganthan, A. Oner, A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion, Appl. Math. Model. 37 (10–11) (2013) 6758–6779.
[8] J. Gao, R. Chen, W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (3) (2013) 641–651.
[9] F. Glover, Tabu search: a tutorial, Interfaces 20 (4) (1990) 74–94.
[10] J. Grabowski, J. Pempera, The permutation flow shop problem with blocking. A tabu search approach, OMEGA 35 (3) (2007) 302–311.
[11] J.N.D. Gupta, Two-stage, hybrid flow shop scheduling problem, J. Oper. Res. Soc. 39 (4) (1988) 359–364.
[12] A. Janiak, E. Kozan, M. Lichtenstein, C. Oguz, Metaheuristic approaches to the hybrid flow shop scheduling problem with a cost-related criterion, Int. J. Prod. Econ. 105 (2) (2007) 407–424.
[13] .Z.H. Jin, K. Ohno, T. Ito, S.E. Elmaghraby, Scheduling hybrid flowshops in printed circuit board assembly lines, Prod. Oper. Manage. 11 (2) (2002) 216–230.
[14] C. Kahraman, O. Enginb, I. Kaya, R.E. Öztürk, Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach, Appl. Soft. Comput. 10 (4) (2010) 1293–1300.
[15] D. Karaboga, An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
[16] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, J. Global. Optim. 39 (3) (2007) 459–471.
[17] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Appl. Soft. Comput. 8 (1) (2008) 687–697.
[18] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Appl. Math. Comput. 214 (1) (2009) 108–132.
[19] X. Lei, J. Tian, L. Ge, A. Zhang, The clustering model and algorithm of PPI network based on propagating mechanism of artificial bee colony, Inform. Sci. 247 (20) (2013) 21–39.
[20] J.Q. Li, Q.K. Pan, Y.C. Liang, An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems, Comput. Ind. Eng. 59 (4) (2010) 647–662.
[21] J.Q. Li, Q.K. Pan, K.Z. Gao, Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems, Int. J. Adv. Manuf. Technol. 55 (9–12) (2011) 1159–1169.
[22] C.J. Liao, E. Tjandradjaja, T.P. Chung, An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem, Appl. Soft. Comput. 12 (6) (2012) 1755–1764.
[23] H.T. Lin, C.J. Liao, A case study in a two-stage hybrid flow shop with setup time and dedicated machines, Int. J. Prod. Econ. 86 (2) (2003) 133–143.
[24] S.W. Lin, K.C. Ying, Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm, OMEGA 41 (2) (2013) 383–389.
[25] B. Liu, L. Wang, Y.H. Jin, An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers, Comput. Oper. Res. 35 (9) (2008) 2791–2806.

[26] F.G. Mohammadi, M.S. Abadeh, Image steganalysis using a bee colony based feature selection algorithm, Eng. Appl. Artif. Intel. 31 (2014) 35–43.
[27] D.C. Montgomery, Design and Analysis of Experiments, John Wiley & Sons, Arizona, 2005.
[28] C. Oguz, M. Ercan, A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks, J. Schedul. 8 (4) (2005) 323–351.
[29] Q.K. Pan, M.F. Tasgetiren, P.N. Suganthan, T.J. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Inform. Sci. 181 (12) (2010) 2455–2468.
[30] Q.K. Pan, L. Wang, L. Gao, W.D. Li, An effective hybrid discrete differential evolution algorithm for the flow shop scheduling with intermediate buffers, Inform. Sci. 181 (3) (2011) 668–685.
[31] 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.
[32] M.C. Portmann, A. Vignier, D. Dardilhac, D. Dezalay, Branch and bound crossed with GA to solve hybrid flowshops, Eur. J. Oper. Res. 107 (2) (1998) 389–400.
[33] 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 (8) (2010) 1439–1454.
[34] R. Ruiz, C. Maroto, A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility, Eur. J. Oper. Res. 169 (3) (2006) 781–800.
[35] R. Ruiz, F.S. Şerifoğlu, T. Urlings, Modeling realistic hybrid flexible flowshop scheduling problems, Comput. Oper. Res. 35 (4) (2008) 1151–1175.
[36] R. Ruiz, J.A. Vázquez Rodríguez, The hybrid flow shop scheduling problem, Eur. J. Oper. Res. 205 (1) (2010) 1–18.
[37] T. Sawik, An exact approach for batch scheduling in flexible flow lines with limited intermediate buffers, Math. Comput. Model. 36 (4) (2002) 461–471.
[38] T. Schiavinotto, T. Stutzle, A review of metrics on permutations for search landscape analysis, Comput. Oper. Res. 34 (10) (2007) 3143–3153.
[39] E. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, Eur. J. Oper. Res. 47 (1) (1990) 65–74.
[40] L.X. Tang, H. Xuan, Lagrangian relaxation algorithms for real-time hybrid flowshop scheduling with finite intermediate buffers, J. Oper. Res. Soc. 57 (3) (2006) 316–324.
[41] H.C. Tsai, Integrating the artificial bee colony and bees algorithm to face constrained optimization problems, Inform. Sci. 258 (10) (2014) 80–93.
[42] L. Wang, L. Zhang, D.Z. Zheng, An effective hybrid genetic algorithm for flow shop scheduling with limited buffers, Comput. Oper. Res. 33 (10) (2006) 2960–2971.
[43] S. Wang, M. Liu, Two-stage hybrid flow shop scheduling with preventive maintenance using multi-objective tabu search method, Int. J. Prod. Res. 52 (5) (2014) 1495–1508.
[44] 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.
[45] B. Wardono, Y. Fathi, A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities, Eur. J. Oper. Res. 155 (2) (2004) 380–401.
[46] H. Xuan, B. Li, Scheduling dynamic hybrid flowshop with serial batching machines, J. Oper. Res. Soc. 64 (6) (2013) 825–832.
[47] 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.
[48] K.C. Ying, S.W. Lin, Multiprocessor task scheduling in multistage hybrid flowshops: an ant colony system approach, Int. J. Prod. Res. 44 (16) (2006) 3161–3177.
[49] M. Zandieh, S.M.T. Fatemi Ghomi, S.M. Moattar Husseini, An immune algorithm approach to hybrid flow shops scheduling with sequence dependent setup times, Appl. Math. Comput. 180 (1) (2006) 111–127.
[50] M. Zandieh, M. Gholami, An immune algorithm for scheduling a hybrid flow shop with sequence-dependent setup times and machines with random breakdowns, Int. J. Prod. Res. 47 (24) (2009) 6999–7027.