Int. J. Appl. Math. Comput. Sci., 2026, Vol. 36, No. 3, 459–474
DOI: 10.61822/amcs-2026-0030

AN IMPROVED TABU SEARCH ALGORITHM FOR PARALLEL TASK SCHEDULING ON IDENTICAL MACHINES

WENHAO JIA[a], XIN CHEN[a], DI WU[b], SHUMIN SHI[a], LIANG ZHAO[c,d,e,*], JEDRZEJ MUSIAL[f], JACEK BLAZEWICZ[f,g]

[a] School of Electronics and Information Engineering
Liaoning University of Technology
Shiying 169, Jinzhou 121001, China

[b] Student Cultural Center, Dalian University of Technology
Linggong 2, Dalian 116024, China

[c] School of Computer and Data Engineering, NingboTech University
Xuefu 1, Ningbo 315104, China
e-mail: liangzhao@nbt.edu.cn

[d] Ningbo Zhoushan Port Group Co., Ltd.
Zhejiang Provincial Seaport Investment & Operation Group Co., Ltd.
Ningdong 269, Ningbo 315042, China

[e] School of Information Science and Engineering
Ningbo University
Fenghua 818, Ningbo 315211, China

[f] Institute of Computing Science & European Center for Bioinformatics and Genomics
Poznan University of Technology
ul. Piotrowo 3a, 60-965 Poznań, Poland

[g] Institute of Bioorganic Chemistry
Polish Academy of Sciences
ul. Noskowskiego 12, 61-704 Poznań, Poland

Aiming at the parallel task scheduling problem ($P|\text{size}_j|C_{\max}$), which requires multiple machines to work collaboratively, this paper proposes an improved tabu search algorithm to overcome the shortcoming of traditional methods being prone to falling into local optima. Firstly, a hybrid neighborhood structure combining swap and insertion operations is designed to enhance search diversity and local optimization capabilities. Secondly, a dynamic tabu length adjustment mechanism is introduced to adaptively change the cycle of tabu objects according to the search process, thereby achieving a balance between breadth and depth of search. Finally, a simplified aspiration criterion is proposed to avoid the loss of high-quality solutions caused by excessive tabu. Extensive computational experiments on randomly generated test instances show that the designed algorithm achieves highly competitive performance compared with the original tabu search algorithm and the OR-Tools solver, while significantly improving computational efficiency. In particular, the proposed method reduces the average $C_{\max}$ by approximately 10.6% on extra-large-scale instances, demonstrating superior solution quality and robustness.

Keywords: parallel task scheduling, identical machine, tabu search, makespan.

1. Introduction

Scheduling problem is one of the main issues in the field of combinatorial optimization, which aims to assign a series of tasks to machines to optimize one or more objectives (Blazewicz *et al.*, 2019), and has a wide range of applications in manufacturing, computer science, logistics and transportation, as well as energy management (Liu *et al.*, 2023; Priyadarshi and Kumar,

*Corresponding author

2025; Wang *et al*., 2020). The classic scheduling model usually assumes that a task is processed by only one machine at any time. However, in modern industrial and computing environments, many tasks require more than one machine or other resources to work collaboratively to be complete (Blazewicz and Weglarz, 1984), and those are called parallel task scheduling problems (Blazewicz *et al*., 1986). For example, in semiconductor manufacturing, a wafer may require multiple devices for collaborative inspection (Chu *et al*., 2024); in the container terminal quay crane scheduling problem, a ship may demand multiple cranes for joint handling (Song *et al*., 2025); in high-performance computing, a large-scale computing job may need to run in parallel on multiple computing nodes (Wei *et al*., 2024).

The problem of parallel task scheduling on identical machines was formally defined by Blazewicz *et al*. (1986), and can be represented by the three filed notation (Graham *et al*., 1979) as $P|\text{size}_j|C_{\max}$, in which P denotes the parallel identical machine environment, size_j indicates the number of machines that task j needs to occupy simultaneously, and $C_{\max}$ represents the optimization objective of minimizing the makespan (the maximum completion time among all tasks).

In their seminal paper, Blazewicz *et al*. (1986) addressed the preemptive and nonpreemptive scenarios involving multiple machine tasks. For the subproblems where each task needs either 1 or k machines, they proposed two polynomial-time algorithms both with a time complexity of $O(n)$. One is designed for the nonpreemptive subproblem, and the other targets the preemptive subproblem. Du and Leung (1989) further analyzed its complexity: when the number of machines $m = 2$ or 3, the problem can be solved by a pseudo-polynomial time algorithm, but when $m \geq 5$, the problem becomes a strongly NP-hard one. In terms of exact algorithms, Błądek *et al*. (2015) proposed two branch-and-bound algorithms for solving contiguous and non-contiguous scheduling problems, but they only perform well on small-scale ones. Kowalczyk and Leus (2018) proposed a branch-and-price method for the parallel task scheduling problem, which can efficiently obtain optimal solutions for medium-scale parallel task scheduling problems with up to 150 jobs and 3–12 machines. Wu and Wang (2018) proposed an improved branch-and-bound algorithm, introduced five efficient lower bounds, and designed three pruning criteria, which significantly improved the solution capability, far exceeding the solution scale of the previous algorithm (Błądek *et al*., 2015). In terms of heuristic algorithms, Switalski and Seredynski (2015) proposed a generalized extremal optimization (GEO) one, which outperforms genetic algorithms on instances with up to 500 jobs and up to 48 machines. Wu and Wang (2018) designed a tabu search (TS) algorithm to find high-quality solutions through neighborhood search of task sequences, providing a reference framework for the heuristic design of similar scheduling problems.

Recently, reinforcement learning-based scheduling methods have attracted increasing attention. Lee *et al*. (2023) proposed a double deep Q-learning framework for parallel dedicated machine scheduling by formulating the problem as a sequential decision-making process. This line of research highlights the potential of learning-based approaches to capture complex system dynamics, while also illustrating an alternative direction complementary to traditional heuristic and meta-heuristic scheduling methods. In terms of heuristic algorithms, Li *et al*. (2024) proposed a hybrid algorithm that integrates K-means clustering and teaching-learning-based optimisation (TLBO). This algorithm outperforms other heuristic approaches, including TLBO and the improved genetic algorithm (IGA) used in the study, in large-scale parallel machine scheduling instances involving up to 500 jobs and 20 machines.

Wu and Wang (2018) propose a tabu search approach in which solutions are represented as permutations of tasks and evaluated using the list scheduling rule to obtain the corresponding makespan. A critical-path-based neighborhood is constructed by tracing the longest path in the disjunctive graph and generating neighboring solutions through task swaps within the associated machine sequences. A tabu mechanism with an aspiration criterion is employed to avoid cycling while allowing improving moves, and the search terminates when predefined stopping conditions are met.

Based on the research of Wu and Wang (2018), this paper proposes an improved tabu search (ITS) algorithm. Firstly, a hybrid neighborhood structure combining swap and insertion operations is designed to enhance the diversity of search and improve the local search ability of the algorithm. Secondly, a dynamic tabu length adjustment mechanism is introduced to adaptively change the cycle of tabu objects according to the search process, to balance the breadth and depth of search. Furthermore, a more efficient aspiration criterion is adopted to ensure that the algorithm will not miss better solutions due to excessive tabu. Finally, through a large number of computational experiments, the superiority of the proposed ITS algorithm over the TS one and the mathematical solver in terms of solution quality and efficiency is verified.

The rest of this paper is organised as follows. Section 2 describes and models the $P|\text{size}_j|C_{\max}$ problem. Section 3 elaborates on the design of the ITS algorithm, Section 4 presents and analyzes the experimental results, while Section 5 summarizes the text and discusses future research directions.

Table 1. Model parameters.

Parameter	Meaning
J	Task set, $J = \{1, 2, \ldots, n\}$, where n is the number of tasks
M	Machine set, $M = \{1, 2, \ldots, m\}$, where m is the number of machines
j, k	Indices of tasks, $j \neq k$
i	Index of machines
p_j	Processing time of task j on the machine
size_j	Number of machines required by task j
S_j	Start time of task j on the machine
C_j	Completion time of task j
$C_{\max}$	Makespan
B	Large real number

Table 2. Model decision variables.

Decision variable	Meaning
$x_{i,j} \in \{0, 1\}$	1 if task j is assigned to machine i, otherwise 0
$y_{i,j,k} \in \{0, 1\}$	1 if the processing of task j on machine i is prior to that of task k, otherwise 0

2. Problem description and modeling

The $P|\text{size}_j|C_{\max}$ problem considered in this study arises in scheduling environments where tasks require multiple machines to operate simultaneously. The key constraints of the problem include the following: (1) each task can only start after all its required machines are idle; (2) the machine assignment must satisfy the task's size_j, which denotes the number of machines required by task j; and (3) the objective is to minimize the makespan $C_{\max}$, defined as the completion time of the last task. The problem can be formally described as below.

Let $J = \{1, 2, \ldots, n\}$ denote the set of tasks, indexed by j and k, and let $M = \{1, 2, \ldots, m\}$ represent the set of identical machines, indexed by i. Each task $j \in J$ has a processing time p_j and requires size_j machines simultaneously ($1 \leq \text{size}_j \leq m$) during its processing. The processing of task j starts at time S_j and occupies the time interval $[S_j, S_j + p_j]$. All machines are available at time zero, and task processing is non-preemptive. The objective is to determine the start times S_j of all tasks so as to minimize the makespan $C_{\max} = \max\{S_j + p_j \mid j \in J\}$.

Based on the above problem description, we now formulate an integer linear programming (ILP) model for the $P|\text{size}_j|C_{\max}$ problem. The model parameters and decision variables are listed in Tables 1 and 2, respectively. The objective function and constraints are as follows:

1. tasks are non-preemptive, and the completion time is equal to the start time plus the processing time;
2. all task completion times are no greater than the makespan, with all task start times taking non-negative values;
3. each task must be assigned exactly to size_j machines;
4. tasks assigned to the same machine are processed sequentially without overlap;
5. any two tasks assigned to the same machine have a well-defined and mutually exclusive processing order;
6. the assignment and sequencing decision variables are of binary nature.

Objective function. We have

$$\min C_{\max}.$$

Constraint functions. We have

$$C_j = S_j + p_j, \quad \forall j \in J, \tag{1}$$

$$C_j \leq C_{\max}, \quad S_j \geq 0, \quad \forall j \in J, \tag{2}$$

$$\sum_{i=1}^{m} x_{i,j} = size_j, \quad \forall j \in J, \tag{3}$$

$$S_j + p_j \leq S_k + (1 - y_{i,j,k})B, \quad \forall j, k \in J, \forall i \in M, \tag{4}$$

$$x_{i,j} + x_{i,k} \leq y_{i,j,k} + y_{i,k,j} + 1, \quad \forall j, k \in J, \forall i \in M, \tag{5.1}$$

$$y_{i,j,k} + y_{i,k,j} \leq 1, \quad \forall j, k \in J, \forall i \in M, \tag{5.2}$$

$$x_{i,j} \in \{0, 1\}, \quad y_{i,j,k} \in \{0, 1\}, \quad \forall j, k \in J, \forall i \in M. \tag{6}$$

3. Improved tabu search algorithm

The tabu search algorithm (Umam *et al.*, 2022) is a meta-heuristic approach that simulates the human memory function. It records and prohibits recently visited solutions or moves using a tabu list, thereby guiding the search process to escape local optima and explore a broader solution space.

The main symbols and parameters used in the proposed ITS algorithm are summarized in Table 3.

Table 3. Main symbols and parameters.

Symbol	Meaning
π	Permutation representing a solution
$\pi^{(t)}$	Current solution at iteration t
π'	Candidate neighboring solution
$N(\pi^{(t)})$	Neighborhood of the current solution $\pi^{(t)}$
π^*	Best solution found so far
$C_{\max}(\pi)$	Makespan of solution π
L	Tabu list
$\|L\|$	Tabu list length
$f(\pi')$	Access frequency of solution π'
$I_{\max}$	Maximum number of iterations
$\|L\|_0$	Initial tabu list length
$\|L\|_{\min}$	Minimum tabu list length
$\|L\|_{\max}$	Maximum tabu list length
λ	Frequency penalty coefficient
K_s	Stagnation threshold
N_c	Maximum neighborhood size

3.1. Algorithm flow. The flow of the improved tabu search algorithm can be divided into the following main stages: data preparation and initialization, iterative optimization, adaptive adjustment, and result statistics and analysis. The specific operations are shown in Fig. 1.

The figure presents the flowchart of the proposed tabu search algorithm, including initialization, neighborhood generation, aspiration-based selection, and adaptive adjustment. The algorithm iteratively explores neighborhood solutions under tabu restrictions and aspiration criteria, while adaptive strategies are applied to alleviate search stagnation. The procedure terminates upon reaching the maximum iteration limit, and the best solution found is reported.

3.1.1. Data preparation stage. Load task data from the dataset and screen out tasks that cannot be scheduled. Initialize the tabu list and the frequency penalty dictionary (used to record the number of accesses to solutions to avoid loops and improve efficiency). When generating the initial solution, first randomly sort the remaining schedulable tasks to generate an initial sorting sequence, use the list scheduling function to calculate the completion time of the initial sequence, and record the current sequence as the optimal solution.

3.1.2. Iterative optimization stage (tabu search stage). Generate the neighborhood of the current solution, evaluate each neighbor solution, select the optimal neighbor solution, update the current solution and the global optimal solution, add the current solution to the tabu list (or grant amnesty if the aspiration condition is met), update the frequency penalty dictionary, check the stagnation state, and record the optimal makespan of this iteration.

3.1.3. Adaptive adjustment stage. This stage mainly includes the dynamic adjustment of the tabu list size and the setting of the frequency penalty mechanism. When stagnation occurs, the tabu list is reduced to increase search diversity. When there is improvement, the tabu list is gradually expanded to strengthen local search. The setting of the penalty mechanism can effectively prevent the algorithm from falling into loops.

The detailed procedure of the proposed ITS algorithm is summarized in Algorithm 1. It is initiated with an initial solution and, subsequently, it explores neighbourhood solutions iteratively under tabu restrictions and aspiration criteria. The incorporation of frequency-based penalties and stagnation detection serves to enhance search diversification. The search is terminated after a predetermined number of iterations, and the optimal solution found is returned.

3.2. Solution representation and evaluation. Consistent with the classic TS algorithm, this paper adopts an indirect coding method based on task sequences. For any given task sequence π, the list scheduling (LS) rule (Squires *et al.*, 2022) is used for decoding to generate a scheduling scheme including specific machine assignments and time arrangements, and the makespan $C_{\max}$ is calculated.

Let r denote the index of a machine ($r = 1, 2, \ldots, m$). The scheduling process proceeds in a sequence of decisions indexed by l. A subset $Y_j \subseteq M$ is selected from all machines, where $|Y_j| = \text{size}_j$, and the machines in Y_j have the earliest available times.

Let A_r^{l-1} denote the availability (completion) time of machine r after the l-1-th scheduling decision. The start time of task j is determined by the latest availability time among the selected machines, which is given by

$$S_j = \max_{r \in Y_j} A_r^{l-1}. \tag{7}$$

After task j starts execution, the availability times of machines are updated. Let p_j denote the processing time of task j. The updated availability time of each machine is defined as

$$A_r^l = \begin{cases} S_j + p_j, & \text{if } r \in Y_j, \\ A_r^{l-1}, & \text{if } r \notin Y_j. \end{cases} \tag{8}$$

Following the list scheduling (LS) rule, tasks are assigned to machines one by one according to the sequence π. By recursively computing the start time S_j and updating the machine availability times A_r^l for all tasks, the makespan $C_{\max}$ of the schedule is finally obtained.

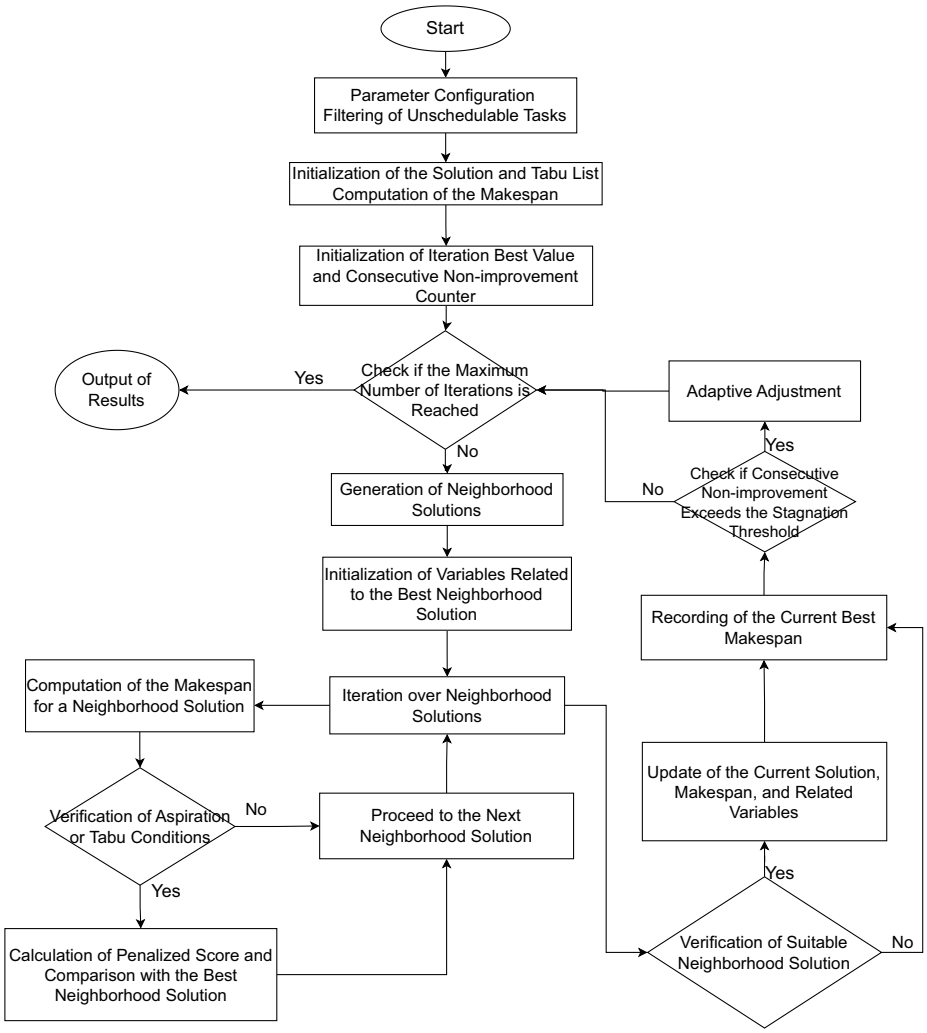


Fig. 1. Flowchart of the improved tabu search algorithm.

3.3. Initial solution generation. A high-quality initial solution helps accelerate the convergence of the algorithm (Blazewicz *et al*., 2008). In the task data loading phase of this paper, special "removed tasks" are first identified and screened according to strict resource constraint conditions. Specifically, if the number of machines required by a task exceeds the difference between the total number of machines in the system and the minimum number of machines required among all tasks, the task is determined as a removed one. Due to their high resource requirements, such tasks cannot be executed simultaneously with any others, and their processing must occupy an exclusive time period. Therefore, the algorithm marks these tasks as removed, directly accumulates their processing time into the final completion time, and excludes them from the subsequent sequence optimization process.

For the remaining schedulable task, an initial scheduling sequence is randomly generated as the starting point of the search to increase the randomness and diversity of the initial solution. The completion time of this sequence is calculated by the list scheduling function, which assigns the size_j machines (where size_j is the number of machines required by the task) with the earliest current availability to each task. The algorithm takes the initial sequence and its completion time as the initial value of the global optimal solution.

3.4. Neighborhood structure. The neighborhood structure (Xie *et al*., 2023) of tabu search is used to define the set of "adjacent" solutions generated from the current task scheduling sequence. These solutions exhibit small differences in relation to the current sequence and are the core exploration objects of local search. In the task scheduling problem, the classic neighborhood generation method is the swap operation: randomly select

two tasks in the sequence and swap their positions. For example, swap the task at positions 1 and 3 in the current sequence $[A, B, C, D]$ to obtain the neighborhood solution $[C, B, A, D]$.

On the basis of the classic swap operation, this paper adopts a hybrid neighborhood structure to enhance the search ability of the algorithm. Specifically, an insertion operation is introduced: the task at position u is removed from the sequence and inserted into position v, where $u \neq v$. All feasible remove–insert position pairs (u, v) are generated and randomly shuffled. To balance search efficiency and neighborhood diversity, neighboring solutions are generated preferentially by swap operations, and insertion operations are used only when the number of generated swap-based neighbors is below the preset maximum neighborhood size. It should be noted that tabu restrictions are not applied during neighborhood generation; instead, all generated neighbors are evaluated first, and the tabu status is checked afterward during the candidate selection step. For example, removing task B from position 2 in the sequence $[A, B, C, D]$ and reinserting it at position 3 yields the new sequence $[A, C, B, D]$. This hybrid structure not only retains the global adjustment capability of the swap operation but also enables local fine-tuning through the insertion operation, thereby improving the diversity of neighborhood solutions and the overall optimization performance of the algorithm.

Taking the $(t-1)$-th iteration as an example, the next solution $\pi^{(t)}$ is selected from the neighborhood $N(\pi^{(t-1)})$. A neighbor solution $\pi' \in N(\pi^{(t-1)})$ can be selected as an acceptable one if and only if it satisfies the following conditions:

$$(\pi' \notin L) \vee (C_{\max}(\pi') < C_{\max}(\pi^*)). \tag{9}$$

Among them, $\pi' \notin L$ is the tabu constraint and $C_{\max}(\pi') < C_{\max}(\pi^*)$ is the aspiration criterion.

3.5. Tabu list and the aspiration criterion. Tabu objects (Rahdar *et al.*, 2022) refer to solutions or operations that are included in the tabu list and temporarily prohibited from being selected by the algorithm. In this paper, the tabu object is set as the complete task sequence, i.e., the neighborhood solution itself. A solution π' is inserted into the tabu list if and only if it does not satisfy the aspiration criterion, namely, $C_{\max}(\pi') \geq C_{\max}(\pi^*)$, and is accepted as the current solution $\pi^{(t)}$. Conversely, any solution with $C_{\max}(\pi') < C_{\max}(\pi^*)$ is exempted from the tabu restriction and directly updates the global best solution $(\pi^*, C_{\max}(\pi^*))$. This mechanism guarantees that no tabu entry dominates the currently known best solution.

The size of the tabu list determines the "memory duration" of tabu objects, and a fixed size is difficult to adapt to complex search environments. For this reason, this paper designs an interval-adaptive tabu list. The length of the list is limited to the basic interval $[10, 50]$ to balance the stability and flexibility of the search. If no better solution is found in 20 consecutive iterations (exceeding the stagnation threshold), it is determined that the algorithm has fallen into a local optimum. At this time, the capacity of the tabu list is reduced (to a minimum of 10), and the current solution is randomly

Algorithm 1. Search procedure of improved tabu search.

Input: Task set J, number of machines m, maximum iterations $I_{\max}$, initial tabu length $|L|_0$, minimum and maximum tabu lengths $|L|_{\min}, |L|_{\max}$, frequency penalty coefficient λ, stagnation threshold K_s, neighborhood size N_c.
Output: Best makespan $C^*_{\max}$ and best sequence π^*.

Generate initial solution $\pi^{(0)}$
$\pi^* \leftarrow \pi^{(0)}$, $C^*_{\max} \leftarrow C_{\max}(\pi^{(0)})$
$L \leftarrow \varnothing$ with $|L| = L_0$, $f(\pi) \leftarrow 0$, $\eta \leftarrow 0$
for $t = 1$ to $I_{\max}$ **do**
  Generate $N(\pi^{(t-1)})$, $|N| \leq N_c$
  $best_score \leftarrow +\infty$, $best_neighbor \leftarrow \varnothing$
  for each $\pi' \in N(\pi^{(t-1)})$ **do**
    Compute $C_{\max}(\pi')$
    if $\pi' \notin L$ **or** $C_{\max}(\pi') < C^*_{\max}$ **then**
      $score \leftarrow C_{\max}(\pi') + \lambda f(\pi')$
      if $score < best_score$ **then**
        $best_neighbor \leftarrow \pi'$, $best_score \leftarrow score$
      end if
    end if
  end for
  if $best_neighbor = \varnothing$ **then**
    continue
  end if
  $\pi^{(t)} \leftarrow best_neighbor$
  if $C_{\max}(\pi^{(t)}) \geq C^*_{\max}$ **then**
    Insert $\pi^{(t)}$ into tabu list L
  end if
  $f(\pi^{(t)}) \leftarrow f(\pi^{(t)}) + 1$
  if $C_{\max}(\pi^{(t)}) < C^*_{\max}$ **then**
    $\pi^* \leftarrow \pi^{(t)}$, $C^*_{\max} \leftarrow C_{\max}(\pi^{(t)})$, $\eta \leftarrow 0$
  else
    $\eta \leftarrow \eta + 1$
  end if
  if $\eta > K_s$ **then**
    Randomly shuffle $\pi^{(t)}$
    $\eta \leftarrow 0$, $|L| \leftarrow, \max(L_{\min}, |L| - 5)$
  else
    $|L| \leftarrow \min(L_{\max}, |L| + 1)$
  end if
end for
return $(C^*_{\max}, \pi^*)$

reset to change the search direction, thereby enhancing the exploration ability of the algorithm. If there is continuous improvement in the search process, the capacity of the tabu list is gradually expanded (to a maximum of 50), and the search breadth is expanded by extending the memory cycle to reduce repeated exploration.

To balance the quality and diversity of solutions, the algorithm adopts an evaluation function that integrates the makespan ($C_{\max}$) and access frequency instead of directly using $C_{\max}$ to evaluate neighbor solutions. Its mathematical expression can be written as

$$\text{Score}(\pi') = C_{\max}(\pi') + \lambda \cdot f(\pi'). \tag{10}$$

Among them, $C_{\max}(\pi')$ is the makespan of solution π', $f(\pi')$ is the historical access frequency of the solution, and λ is the frequency penalty coefficient. The algorithm selects the solution with the smallest evaluation score from all candidate ones that meet the acceptance conditions as the current solution $\pi^{(t)}$ for the next iteration. The calculation formula is as follows:

$$\pi^{(t)} = \arg\min \Big\{ \text{Score}(\pi') \mid \pi' \in N(\pi^{(t-1)}) \wedge ((\pi' \notin L) \vee (C_{\max}(\pi') < C_{\max}(\pi^*))) \Big\}. \tag{11}$$

The aspiration criterion is a key manifestation of the algorithm's flexibility. It allows tabu restrictions to be overridden when a candidate move produces a solution of superior quality. In the proposed algorithm, a neighborhood solution π' may be accepted even if its sequence is contained in the tabu list L, provided that $C_{\max}(\pi') < C_{\max}(\pi^*)$. Since the list scheduling rule is deterministic, the makespan corresponding to any given task sequence is uniquely defined. Therefore, a previously visited solution cannot subsequently produce a makespan smaller than the best value obtained so far. In this deterministic setting, the aspiration condition is theoretically redundant. Nevertheless, it is retained for completeness and consistency with the standard tabu search framework. In practice, it primarily guarantees that any genuinely superior solution—necessarily a newly generated sequence—is always accepted. This design preserves the theoretical generality of the algorithm without affecting its behavior under deterministic evaluation.

The above design effectively improves the optimization ability and robustness of tabu search in task scheduling problems through the precise definition of tabu objects, the adaptive adjustment of the dynamic tabu list, the evaluation function with penalty, and the flexible aspiration criterion.

Table 4. Experimental parameter settings.

Name	Value
Maximum number of iterations	100
Initial tabu list length	20
Stagnation threshold	20
Maximum tabu list length	50
Maximum neighborhood size	50
Minimum tabu list length	10

4. Numerical experiments and result analysis

To comprehensively evaluate the performance of the ITS algorithm, a series of numerical experiments were designed in this paper. The ITS algorithm was compared with the original TS one and the mathematical programming solver OR-Tools from three aspects: solution quality, stability, and computational efficiency. All algorithms were implemented using Python 3.13 and run on the PyCharm 2021 platform. The experimental environment was a computer equipped with an Intel(R) Core(TM) i9-14900KF 3.20 GHz processor and 64 GB of memory.

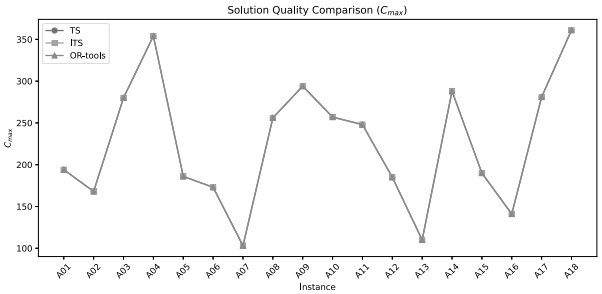
4.1. Experimental data. The instance generation method by Wu and Wang (2018) was adopted. According to the task scale, the experiments were divided into five parts (A, B, C, D, E), containing 18, 30, 15, 40, and 30 instances respectively. For parts A and B of the experiments, the OR-Tools mathematical solver, the TS algorithm, and the ITS one were used for solving. For parts C, D, and E, only the TS and ITS algorithms were employed. The focus was on the makespan, standard deviation, and runtime of each method.

The number of tasks $n \in [5, 500]$ and the number of machines $m \in [3, 50]$ were selected. For each combination of (n, m), one instance was randomly generated (each experiment was repeated 30 times to eliminate the impact of randomness). The task sequence number of each instance was randomly shuffled, and, for each task; the required number of machines size_j was randomly selected from $1, \ldots, m$, while its processing time p_j was randomly selected from $1, \ldots, 80$.

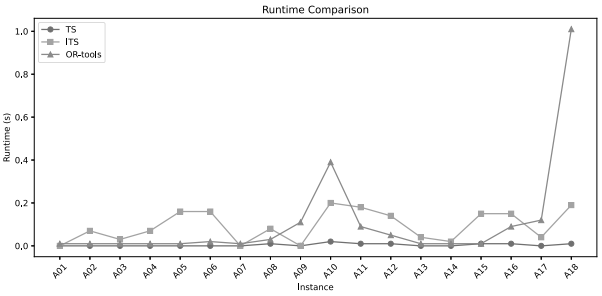
4.2. Comparative algorithms and parameter settings. The comparative algorithms included: ITS (the improved tabu search algorithm proposed in this paper), TS (the tabu search algorithm proposed by Wu and Wang (2018)), and OR-Tools (a mathematical programming solver developed by Google, with a time limit set to 3600 seconds). The parameter settings are shown in Table 4. The main ones include: the maximum number of iterations of 100, the initial tabu list length of 20, the tabu list length range of

Table 5. Small-scale instances (Part A).

Instance	m	n	TS			ITS			OR-Tools		
			TS_Cmax	TS_std	TS_Time	ITS_Cmax	ITS_std	ITS_Time	OR_Cmax	OR_std	OR_Time
A01	3	5	194	0	0	194	0	0	194	0	0.01
A02	3	6	168	0	0	168	0	0.07	168	0	0.01
A03	3	7	280	0	0	280	0	0.03	280	0	0.01
A04	3	8	354	0	0	354	0	0.07	354	0	0.01
A05	3	9	186	0	0	186	0	0.16	186	0	0.01
A06	3	10	173	0	0	173	0	0.16	173	0	0.02
A07	4	5	103	0	0	103	0	0	103	0	0.01
A08	4	6	256	0	0.01	256	0	0.08	256	0	0.03
A09	4	7	294	0	0	294	0	0	294	0	0.11
A10	4	8	257	0	0.02	257	0	0.2	257	0	0.39
A11	4	9	248	0	0.01	248	0	0.18	248	0	0.09
A12	4	10	185	0	0.01	185	0	0.14	185	0	0.05
A13	5	5	110	0	0	110	0	0.04	110	0	0.01
A14	5	6	288	0	0	288	0	0.02	288	0	0.01
A15	5	7	190	0	0.01	190	0	0.15	190	0	0.01
A16	5	8	141	0	0.01	141	0	0.15	141	0	0.09
A17	5	9	281	0	0	281	0	0.04	281	0	0.12
A18	5	10	361	0	0.01	361	0	0.19	361	0	1.01
Average			226.06	0	0	226.06	0	0.09	226.06	0	0.11

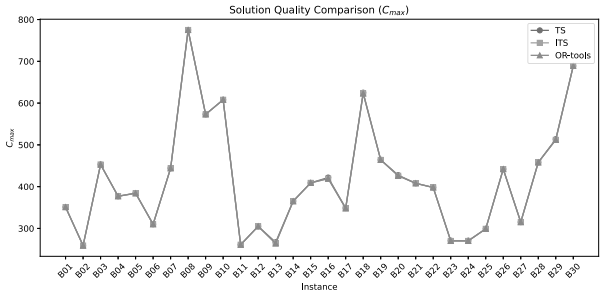


(a) comparison of $C_{\max}$ on small-scale instances.

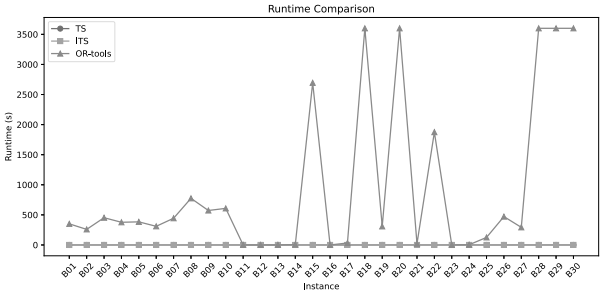


(b) runtime comparison on small-scale instances.

Fig. 2. Performance comparison on small-scale instances.



(a) comparison of $C_{\max}$ on small-medium-scale instances.



(b) runtime comparison on small-medium-scale instances.

Fig. 3. Performance comparison on small-medium-scale instances.

Table 6. Small and medium-scale instances (Part B).

Instance	m	n	TS			ITS			OR-Tools		
			TS_Cmax	TS_std	TS_Time	ITS_Cmax	ITS_std	ITS_Time	OR_Cmax	OR_std	OR_Time
B01	3	11	351	0	0.01	351	0	0.13	351	0	351
B02	3	12	259	0	0.03	259	0	0.2	259	0	259
B03	3	13	453	0	0.02	453	0	0.2	453	0	453
B04	3	14	377	0	0.02	377	0	0.18	377	0	377
B05	3	15	384.7	0.48	0.05	384	0	0.24	384	0	384
B06	3	16	310	0	0.03	310	0	0.22	310	0	310
B07	3	17	444	0	0.06	444	0	0.26	444	0	444
B08	3	18	775	0	0.03	775	0	0.21	775	0	775
B09	3	19	573	0	0.06	573	0	0.26	573	0	573
B10	3	20	608	0	0.12	608	0	0.34	608	0	608
B11	4	11	261	0	0.03	261	0	0.22	261	0	0.55
B12	4	12	305	0	0.03	305	0	0.22	305	0	0.06
B13	4	13	266.9	5.04	0.04	264	0	0.25	264	0	0.48
B14	4	14	365.5	1.58	0.07	365	0	0.29	365	0	5.65
B15	4	15	409	0	0.07	409	0	0.29	409	0	2694.71
B16	4	16	421.8	3.36	0.05	419.2	0.42	0.27	419.2	0.42	0.04
B17	4	17	348.8	2.2	0.13	348	0	0.36	348	0	33.01
B18	4	18	625	2.11	0.05	623	0	0.27	623	0	3600.18
B19	4	19	464.3	0.48	0.12	464	0	0.36	464	0	308.68
B20	4	20	427.5	1.78	0.15	426	0	0.39	426	0	3600.17
B21	5	11	408	0	0.03	408	0	0.24	408	0	20.59
B22	5	12	398	0	0.06	398	0	0.29	398	0	1874.76
B23	5	13	270.8	0.42	0.03	270	0	0.23	270	0	0.02
B24	5	14	270.6	0.97	0.07	270	0	0.31	270	0	3.82
B25	5	15	299	0	0.05	299	0	0.28	299	0	125.43
B26	5	16	442	0	0.05	442	0	0.28	442	0	471.48
B27	5	17	315.1	0.32	0.11	315	0	0.36	315	0	292.9
B28	5	18	458	0	0.14	458	0	0.39	458	0	3600.22
B29	5	19	513.6	3.69	0.17	512	0	0.43	512	0	3600.15
B30	5	20	689	0	0.17	689	0	0.43	689	0	3600.25
Average			416.45	0.75	0.07	415.97	0.01	0.28	415.97	0.01	945.57

$[10, 50]$, the stagnation threshold of 20, and the maximum neighborhood size of 50. The penalty coefficient was set to 0.1, and each instance was run repeatedly 30 times.

The parameter settings used in this study were determined based on preliminary experiments. In particular, the penalty coefficient λ and the tabu list length range were selected after testing multiple candidate values to achieve a balance between solution quality and computational efficiency. Similarly, the maximum number of iterations and the stagnation threshold were empirically determined to ensure sufficient search depth while avoiding excessive computational cost and premature stagnation. It is noteworthy that analogous parameter ranges and settings have been extensively adopted in related tabu search studies.

The algorithm performance was evaluated using the following indicators: Cmax (makespan, measuring the overall quality of solutions), std (standard deviation of makespan, reflecting the stability of solutions), and Time (average runtime in seconds, measuring computational efficiency). Later we will use $Algo_I$ to describe the indicator I of the method $Algo$, in which $Algo \in \{$TS, ITS, OR-Tools$\}$ and $I \in \{$Cmax, std, Time$\}$.

4.3. Experimental results and analysis. The proposed tabu search algorithm incorporates stochastic elements, such as random initialization and neighborhood sampling. To enhance reproducibility and ensure fair comparisons, all experimental runs were performed under controlled random settings. In particular, each problem instance was associated with a predefined base random seed, while multiple runs were carried out using systematic variations of this seed. This design enables full reproducibility of the experimental results while preserving the stochastic nature of the algorithm. Accordingly, the standard deviations reported in Tables

4–8 represent performance variability under controlled randomness rather than arbitrary fluctuations.

4.3.1. Small-scale instances (Part A). In Part A, the ITS, TS and OR-Tools algorithms all obtained the same optimal solutions in all instances, with a standard deviation of zero (Table 5). This indicates that all three methods can stably find the optimal solution for small-scale problems. To provide a more intuitive visualization of the overall performance trends on small-scale instances, Fig. 2 presents the line chart comparisons of TS, ITS, and OR-Tools in terms of makespan and runtime.

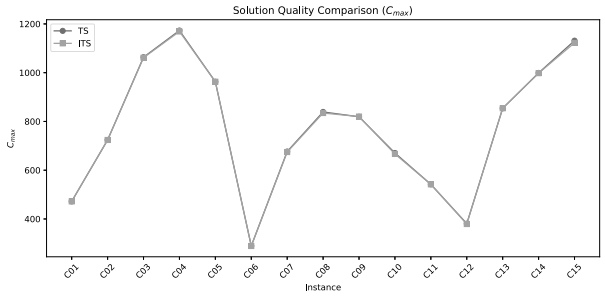
4.3.2. Small-medium-scale instances (Part B). In Part B, the $C_{\max}$ of ITS was lower than that of TS in 10 instances (e.g., B05, B13, B18), and was consistent with the optimal solution of OR-Tools (e.g., ITS_Cmax = 264.00 = OR_Cmax in B13), indicating that the ITS algorithm began to demonstrate its optimization capability. The average standard deviation of the makespan of ITS was 0.01, i.e., significantly lower than 0.75 of TS (e.g., TS_std = 5.04 and ITS_std = 0 in B13), which shows that the search process of the improved algorithm is more stable.

OR-Tools timed out in some instances (e.g., runtime exceeded 3600 seconds in B18 and B20), while the average runtime of ITS (0.28 seconds) was only four times that of TS (TS = 0.07 seconds), but this trade-off resulted in solutions of higher quality. To further examine the performance trends as the problem scale increases, Fig. 3 extends the comparison to small and medium-scale instances, illustrating the makespan and runtime results of TS, ITS, and OR-Tools.

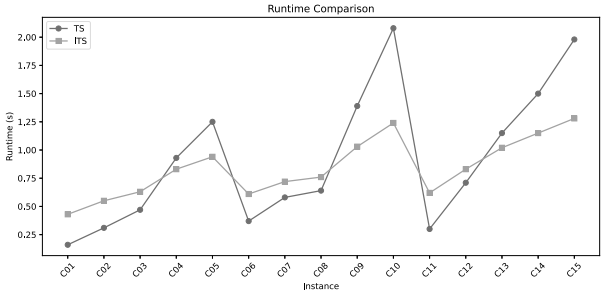
4.3.3. Medium-scale instances (Part C). In terms of solution quality, the average $C_{\max}$ of ITS (771.32) was lower than that of TS (773.15), with consistent improvements observed in instances such as C03 and C04 (e.g., 1169 vs 1173). The average standard deviation of ITS (1.04) was also smaller than that of TS (2.99), indicating better stability.

The average runtime of ITS (0.84 seconds) was slightly lower than that of TS (0.92 seconds), while ITS also achieved improved solution quality and stability. This demonstrates that the ITS algorithm is both efficient and effective for medium-scale problem instances. To further highlight the performance differences between TS and ITS on medium-scale instances, Fig. 4 illustrates their makespan and runtime trends using line chart comparisons.

4.3.4. Large-scale instances (Part D). For large-scale instances, the advantage of the ITS algorithm became more obvious. The average $C_{\max}$ of ITS (2376.52) was much lower than that of TS (2398.53), with improvements in almost all instances.



(a) comparison of $C_{\max}$ on medium-scale instances.



(b) runtime comparison on medium-scale instances.

Fig. 4. Performance comparison on medium-scale instances.

The average standard deviation of ITS (21.82) was also smaller than that of TS (24.01), which proved that the ITS algorithm had better stability in dealing with large-scale complex problems. Furthermore, the average runtime of ITS (7.33 seconds) was lower than that of TS (46.65 seconds), which was a reasonable cost for better solution quality. To assess the scalability of TS and ITS on large-scale instances, Fig. 5 depicts their makespan and runtime performance through line chart comparisons.

4.3.5. Extra-large-scale instances (Part E). In extra-large-scale instances, the performance of the ITS algorithm was outstanding. The average $C_{\max}$ of ITS (8454.57) was far lower than that of TS (9454.95), and the improvement range was more obvious. For example, in E08, ITS reduced $C_{\max}$ from 7355.8 of TS to 6648.

The standard deviation of ITS (53.21) was also better than that of TS (101.41), which showed that the ITS algorithm could maintain good stability in the face of extremely complex problems. Besides, the average runtime of ITS (60.99 seconds) was lower than that of TS (249.9 seconds), and the significant improvement in solution quality made this time cost acceptable. To evaluate the robustness of TS and ITS under the most

Table 7. Medium-scale instances (Part C).

Instance	m	n	TS			ITS		
			TS_Cmax	TS_std	TS_Time	ITS_Cmax	ITS_std	ITS_Time
C01	6	20	472	0	0.16	472	0	0.43
C02	6	25	724	0	0.31	724	0	0.55
C03	6	30	1063.4	3.57	0.47	1061.4	0.52	0.63
C04	6	35	1173	3.09	0.93	1169	0	0.83
C05	6	40	962.9	3.14	1.25	962.9	2.85	0.94
C06	8	20	289.3	0.48	0.37	289.1	0.32	0.61
C07	8	25	676.7	2.45	0.58	674.5	1.58	0.72
C08	8	30	839	5.98	0.64	834.5	1.9	0.76
C09	8	35	819.1	3	1.39	820.1	3.51	1.03
C10	8	40	670.6	4.86	2.08	666.7	1.77	1.24
C11	10	20	542	0	0.3	542	0	0.62
C12	10	25	380.9	2.88	0.71	379.2	0.42	0.83
C13	10	30	854	0	1.15	854	0	1.02
C14	10	35	999	3.16	1.5	998	0	1.15
C15	10	40	1131.4	12.29	1.98	1122.4	2.76	1.28
Average			773.15	2.99	0.92	771.32	1.04	0.84

challenging extra-large scale instances, Fig. 6 illustrates their makespan and runtime performance trends.

4.4. Comprehensive analysis. From the experimental results of different scales, the following conclusions can be drawn.

Solution quality. The ITS algorithm proposed in this paper has obvious advantages in the solution quality compared with the original TS algorithm. With the increase of the problem scale, the improvement effect of ITS becomes more and more significant, especially in large-scale and extra-large-scale instances.

Stability. The standard deviation of the ITS algorithm is smaller than that of TS in all scale instances, which shows that ITS has better stability and can obtain more consistent solutions in repeated runs.

Computational efficiency. The ITS algorithm not only achieves shorter runtime than TS, but also provides significantly better solution quality and stability, showing excellent comprehensive performance.

Comparison with OR-Tools. For small-scale instances, OR-Tools can obtain the optimal solution, but for medium-scale and above instances, OR-Tools often times out and cannot obtain a solution, while the ITS algorithm can still obtain high-quality solutions within an acceptable time.

In general, the ITS algorithm proposed in this paper exhibits better performance in solving the parallel task scheduling problem with machine constraints, and is of good application value in practical engineering.

5. Conclusion

This paper focused on the parallel task scheduling problem with multiple machines ($P|\text{size}_j|C_{max}$) and aimed to enhance the performance of the existing tabu search algorithm in solving large-scale complex instances. An ITS algorithm was proposed. To address the issues of the existing tabu search solution, such as being prone to falling into local optima and having low search efficiency, this study improved the algorithm mainly from three aspects. Firstly, a hybrid neighborhood structure was designed, combining insertion and swap operations to enhance the search ability of the algorithm. Secondly, a dynamic tabu length was introduced, allowing the length of the tabu list to be adaptively adjusted within a set interval to balance the depth and breadth of the search. Thirdly, the aspiration criterion was optimized to simplify the complexity of operations and improve the flexibility of the algorithm.

The numerical experiment results show that the ITS algorithm performs excellently. In small-scale instances (Part A), the solution quality of ITS is consistent with that of the original tabu search algorithm and the mathematical solver OR-Tools. In small-medium-scale instances (Part B), the average makespan of ITS is slightly better than that of TS, and its stability is also better. In medium-scale (Part C), large-scale (Part D), and extra-large-scale (part E) instances, the advantages of ITS are particularly significant. It has a smaller average makespan and a lower standard deviation (indicating that the algorithm is more stable), and the computational efficiency is greatly improved, with the runtime being much lower than that of TS. Compared with OR-Tools, ITS can find better solutions faster for large-scale problems.

Table 8. Large-scale instances (Part D).

Instance	m	n	TS			ITS		
			TS_Cmax	TS_std	TS_Time	ITS_Cmax	ITS_std	ITS_Time
D01	15	60	1560.5	17.39	11.19	1555	0	2.8
D02	15	70	1678.9	16.1	16.93	1678.3	11.53	3.62
D03	15	80	1649.3	20.45	23.74	1653.1	9.88	4.13
D04	15	90	2128.7	20.74	28.69	2120.1	22.32	4.87
D05	15	100	2042.6	11.37	29.02	2043.5	14.86	4.88
D06	15	110	2422.1	24.29	31.15	2381.3	25.69	6.07
D07	15	120	2874.7	20.42	38.61	2828.2	17.22	6.95
D08	15	130	2759.5	30.98	43.88	2713.7	28.52	7.52
D09	15	140	3222.1	43.68	51.19	3188.4	43.29	8.36
D10	15	150	3588.2	38.12	55.45	3539.2	43.38	8.77
D11	20	60	1271.5	24.65	13.91	1259.4	16.64	3.37
D12	20	70	1830.8	14.74	17.44	1818.5	13.52	3.85
D13	20	80	1908.6	28.12	27.42	1899.4	11.05	4.85
D14	20	90	2169.6	20.91	33.62	2169.6	12.41	5.41
D15	20	100	2313.7	27.64	47.75	2302	22.39	6.33
D16	20	110	2898.8	30.97	32.71	2867.1	29.39	6.44
D17	20	120	2649.1	16	44.56	2626.5	23.97	7.71
D18	20	130	3265.1	44.77	55.85	3223.6	27.34	8.82
D19	20	140	3170.7	26.47	67.56	3126	32.95	9.86
D20	20	150	3407.4	21.65	69.21	3356.8	42.03	10.1
D21	25	60	1520.7	7.75	15.79	1517	0	3.85
D22	25	70	1296	17.88	24.84	1291.7	14.01	4.73
D23	25	80	1622.1	16.82	36.16	1618	22.69	5.79
D24	25	90	1966.9	23.3	46.8	1973.5	17.51	6.71
D25	25	100	2154.8	21.34	59.5	2139.5	26.23	7.57
D26	25	110	3143.3	21.11	48.12	3125.7	25.55	8.32
D27	25	120	2490	39.31	57.85	2486.9	25.75	9.24
D28	25	130	2951.8	23.2	68.33	2883.5	25.26	10.19
D29	25	140	2944.8	29.17	80.42	2884.9	29.18	11.3
D30	25	150	3600.1	25.79	88.82	3573.2	26.45	12.02
D31	30	60	1384.3	12.18	17.61	1369.6	16.67	4.34
D32	30	70	1342.4	12.63	27.73	1337.1	10.69	5.35
D33	30	80	1683.6	16.18	36.62	1683.6	23.33	6.18
D34	30	90	2409.8	22.7	52.81	2390.5	17.25	7.51
D35	30	100	2086.6	16.26	70.29	2098.1	20.77	8.36
D36	30	110	2864.1	37.8	54.3	2804.1	26.1	9.21
D37	30	120	2310.5	21.87	66.85	2286.8	21.35	10.38
D38	30	130	2956.2	27.79	74.54	2932.4	25.37	11.02
D39	30	140	3242.7	29.85	93.24	3195	19.61	12.62
D40	30	150	3158.7	38	105.42	3120	30.45	13.64
Average			2398.53	24.01	46.65	2376.52	21.82	7.33

In summary, the improved tabu search algorithm proposed in this paper is an effective method for solving parallel task scheduling problems, especially suitable for handling large-scale and extra-large-scale complex instances, and can provide higher-quality decision support for actual production scheduling scenarios.

Based on the research results of this paper, future work can be further explored in the following directions:

1. *Enhancement of scalability through computational means.* While ITS exhibits competitive performance, further investigation is necessary to enhance its computational scalability for problem instances of considerable size. Potential avenues for exploration include parallel implementation, adaptive stopping criteria, and surrogate-assisted evaluation, with the objective of reducing runtime

Table 9. Extra-large-scale instances (Part E).

Instance	m	n	TS			ITS		
			TS_Cmax	TS_std	TS_Time	ITS_Cmax	ITS_std	ITS_Time
E01	30	250	6171.6	104.61	83.66	5611.8	39.35	26.58
E02	30	300	7942.9	94.1	126.3	7242.4	47.74	35.84
E03	30	350	8875.7	99	173.92	7842.8	16.89	47.09
E04	30	400	9611.2	100.98	219.17	8660.2	67.58	57.48
E05	30	450	11372.8	106.31	285.5	10056.8	48.94	48.94
E06	30	500	11918.5	130.64	336.79	10670.8	80.44	87.74
E07	35	250	6238.6	69.24	100.616	5645.6	43.76	29.29
E08	35	300	7355.8	77.13	133.21	6648	47.66	38.87
E09	35	350	9228.8	65.25	181.61	8251.8	84.23	49.55
E10	35	400	9432.9	145.46	234.64	8321.4	77.79	62.45
E11	35	450	11228.5	81.06	283.18	10032.8	58.74	73.61
E12	35	500	13092.4	98	362.81	11694.8	27.7	91.42
E13	40	250	6795.6	86.42	99.22	6214.8	50.31	31.16
E14	40	300	7052.2	83.64	149.43	6270.8	44.95	41.75
E15	40	350	8367.4	88.28	203.85	7531.2	35.9	53.04
E16	40	400	10010.2	114.37	262.31	8915	51.53	65.63
E17	40	450	11474.1	108.93	334.21	10114	35.59	81.83
E18	40	500	12577.8	100.47	407.06	11214	31.77	96.92
E19	45	250	6629.2	76.99	114.56	5980	71.76	34.44
E20	45	300	7772.6	107.92	167.8	6990.6	41.31	44.88
E21	45	350	8505.5	110.54	229.3	7500.6	39.75	56.8
E22	45	400	10098.9	99.56	304.57	9054.4	104.29	70.63
E23	45	450	10932.8	176.38	382.79	9696.2	52.55	88.04
E24	45	500	12572.5	149.19	458.33	11190.8	68.58	103.8
E25	50	250	6025.2	88.67	119.67	5505.6	51.41	35.23
E26	50	300	7319.1	74.79	266	6581.8	28.54	47.62
E27	50	350	9494.5	82.22	247.96	8445.2	66.16	63.15
E28	50	400	10247.2	107.74	324.08	9090.2	46.96	73.5
E29	50	450	11856.8	108.44	408.56	10605.4	46.77	88.25
E30	50	500	13447.3	105.82	496.03	12057.4	87.39	104.06
Average			9454.95	101.41	249.9	8454.57	53.21	60.99

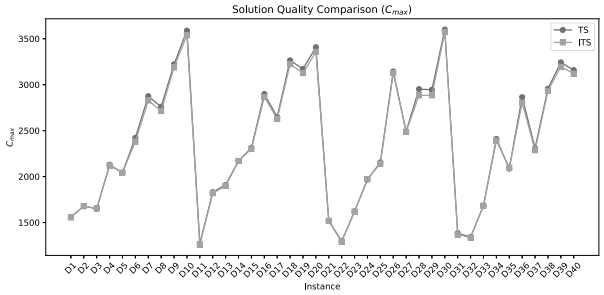
overhead while maintaining solution robustness.

2. *Fusion with other meta-heuristic algorithms.* Explore the fusion of ITS with meta-heuristic algorithms such as the genetic algorithm (Katoch *et al*., 2021), simulated annealing (Kumar *et al*., 2024), and particle swarm optimization (Gad, 2022) to build a hybrid optimization framework, combining the advantages of different algorithms to achieve better performance.

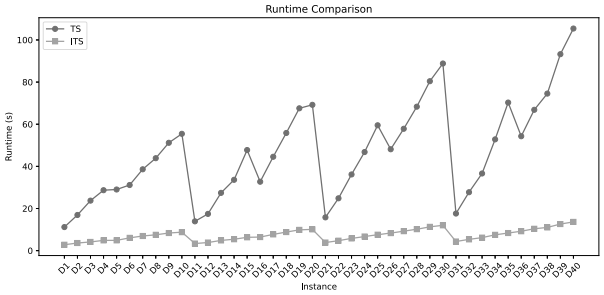
3. *Attempt to use other types of methods to solve the parallel task scheduling problem*, such as the branch-and-bound algorithm (Swat *et al*., 2025), which may provide exact refinement or bounding strategies, as well as machine learning methods (Sarker, 2021) or reinforcement learning approaches (Liu *et al*., 2025), which could be integrated to guide neighborhood selection or adaptive parameter control, thereby further improving the efficiency and robustness of ITS on large-scale instances.

4. *Application and expansion for port container terminal scheduling scenarios.* The algorithm framework proposed in this study has good generality and scalability, and future research can focus on exploring its specific application in port container terminal scheduling. In this scenario, ships are equivalent to tasks, and quay cranes (Rodrigues and Agra, 2022) to parallel machines. A large ship usually requires multiple quay cranes to serve simultaneously (i.e., parallel task), and its operation time depends on the quay crane with the latest completion time, which is a typical $P|\text{size}_j|C_{\max}$ problem.

Future research can use the ITS algorithm proposed in this paper as the core engine, integrate

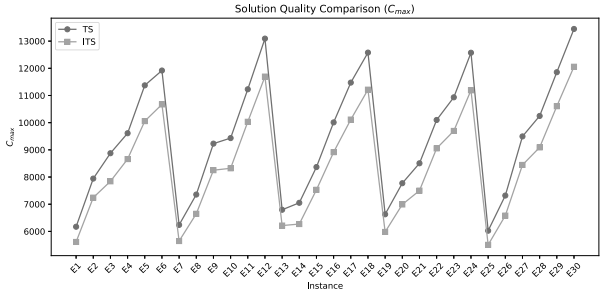


(a) comparison of $C_{\max}$ on large-scale instances.

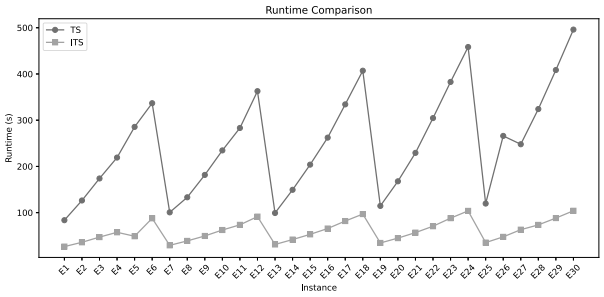


(b) runtime comparison on large-scale instances.

Fig. 5. Performance comparison on large-scale instances.



(a) comparison of $C_{\max}$ on extra-large-scale instances.



(b) runtime comparison on extra-large-scale instances.

Fig. 6. Performance comparison on extra-large-scale instances.

port-specific constraints and optimization objectives (such as minimizing ship time in port, maximizing quay crane utilization, balancing yard operations, etc.), design targeted encoding, decoding methods, and neighborhood operations, and develop an intelligent scheduling system suitable for port environments.

Acknowledgment

The work is supported in part by grants from the National Natural Science Foundation of China (nos. 12371363, W2521074), the Liaoning Revitalization Talents Program (no. XLYC2403125), the Ningbo Science and Technology Innovation 2025 Major Project (no. 2022Z228), and the CAS President's International Fellowship Initiative, PIFI (no. 2025PG0005).

Wenhao Jia sincerely appreciates Jianjun Ge for his help with algorithm implementation and experiment design.

References

Błądek, I., Drozdowski, M., Guinand, F. and Schepler, X. (2015). On contiguous and non-contiguous parallel task scheduling, *Journal of Scheduling* **18**(5): 487–495.

Blazewicz, J., Drabowski, M. and Weglarz, J. (1986). Scheduling multiprocessor tasks to minimize schedule length, *IEEE Transactions on Computers* **C-35**(5): 389–393.

Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G. and Weglarz, J. (2019). *Handbook on Scheduling*, Springer, Cham.

Blazewicz, J., Pesch, E., Sterna, M. and Werner, F. (2008). Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date, *Computers & Operations Research* **35**(2): 574–599.

Blazewicz, J. and Weglarz, J. (1984). Scheduling independent 2-processor tasks to minimize schedule length, *Information Processing Letters* **18**(5): 267–273.

Chu, C.-H., Weng, C.-Y. and Chen, Y.-T. (2024). Enhancing manual inspection in semiconductor manufacturing with integrated augmented reality solutions, *Journal of Manufacturing Systems* **77**: 933–945.

Du, J. and Leung, J.Y.-T. (1989). Complexity of scheduling parallel task systems, *SIAM Journal on Discrete Mathematics* **2**(4): 473–487.

Gad, A.G. (2022). Particle swarm optimization algorithm and its applications: A systematic review, *Archives of Computational Methods in Engineering* **29**(5): 2531–2561.

Graham, R.L., Lawler, E.L., Lenstra, J.K. and Kan, A.R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey, *Annals of Discrete Mathematics* **5**: 287–326.

Katoch, S., Chauhan, S.S. and Kumar, V. (2021). A review on genetic algorithm: Past, present, and future, *Multimedia Tools and Applications* **80**(5): 8091–8126.

Kowalczyk, D. and Leus, R. (2018). A branch-and-price algorithm for parallel machine scheduling using ZDDS and generic branching, *INFORMS Journal on Computing* **30**(4): 768–782.

Kumar, B.A., Jyothi, B., Singh, A.R., Bajaj, M., Rathore, R.S. and Tuka, M.B. (2024). Hybrid genetic algorithm-simulated annealing based electric vehicle charging station placement for optimizing distribution network resilience, *Scientific Reports* **14**(1): 7637.

Lee, D., Kang, H., Lee, D., Lee, J. and Kim, K. (2023). Deep reinforcement learning-based scheduler on parallel dedicated machine scheduling problem towards minimizing total tardiness, *Sustainability* **15**(4): 2920.

Li, Y., Liu, J., Wang, L., Liu, J., Tang, H., Guo, J. and Xu, W. (2024). A k-means-teaching learning based optimization algorithm for parallel machine scheduling problem, *Applied Soft Computing* **161**: 111746.

Liu, X., Chen, X., Chau, V., Musial, J. and Blazewicz, J. (2025). Flexible job shop scheduling problem using graph neural networks and reinforcement learning, *Computers & Operations Research* **182**: 107139.

Liu, X., Wang, W., Chen, X., Sterna, M. and Blazewicz, J. (2023). Exact approaches to late work scheduling on unrelated machines, *International Journal of Applied Mathematics and Computer Science* **33**(2): 285–295, DOI: 10.34768/amcs-2023-0021.

Priyadarshi, R. and Kumar, R.R. (2025). Evolution of swarm intelligence: A systematic review of particle swarm and ant colony optimization approaches in modern research, *Archives of Computational Methods in Engineering* **32**(6): 3609–3650.

Rahdar, S., Ghanbari, R. and Ghorbani-Moghadam, K. (2022). Tabu search and variable neighborhood search algorithms for solving interval bus terminal location problem, *Applied Soft Computing* **116**: 108367.

Rodrigues, F. and Agra, A. (2022). Berth allocation and quay crane assignment/scheduling problem under uncertainty: A survey, *European Journal of Operational Research* **303**(2): 501–524.

Sarker, I.H. (2021). Machine learning: Algorithms, real-world applications and research directions, *SN Computer Science* **2**(3): 160.

Song, Y., Ji, B. and Yu, S.S. (2025). Multi-port berth allocation and time-invariant quay crane assignment problem with speed optimization, *IEEE Transactions on Intelligent Transportation Systems* **26**(1): 128–142.

Squires, M., Tao, X., Elangovan, S., Gururajan, R., Zhou, X. and Acharya, U. R. (2022). A novel genetic algorithm based system for the scheduling of medical treatments, *Expert Systems with Applications* **195**: 116464.

Swat, S., Antczak, M., Zok, T., Blazewicz, J. and Musial, J. (2025). Exact algorithms for the satellite image selection problem, *International Journal of Applied Mathematics and Computer Science* **35**(2): 293–309, DOI: 10.61822/amcs-2025-0021.

Switalski, P. and Seredynski, F. (2015). Scheduling parallel batch jobs in grids with evolutionary metaheuristics, *Journal of Scheduling* **18**(4): 345–357.

Umam, M.S., Mustafid, M. and Suryono, S. (2022). A hybrid genetic algorithm and tabu search for minimizing makespan in flow shop scheduling problem, *Journal of King Saud University Computer and Information Sciences* **34**(9): 7459–7467.

Wang, W., Chen, X., Musial, J. and Blazewicz, J. (2020). Two meta-heuristic algorithms for scheduling on unrelated machines with the late work criterion, *International Journal of Applied Mathematics and Computer Science* **30**(3): 573–584, DOI: 10.34768/amcs-2020-0030.

Wei, X., Wei, H., Lu, M., Zhang, Z., Chang, S., Zeng, S., Wang, F. and Chen, M. (2024). Large-scale parallel embedded computing with improved-MPI in off-chip distributed clusters, *IEEE Transactions on Industrial Informatics* **20**(11): 12575–12585.

Wu, L. and Wang, S. (2018). Exact and heuristic methods to solve the parallel machine scheduling problem with multi-processor tasks, *International Journal of Production Economics* **201**: 26–40.

Xie, J., Li, X., Gao, L. and Gui, L. (2023). A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems, *Journal of Manufacturing Systems* **71**: 82–94.



Wenhao Jia is a second-year postgraduate student at the Liaoning University of Technology, and his main interests and research directions focus on solving multi-objective scheduling problems using reinforcement learning.



Xin Chen received his BE degree in computer science from the Dalian University of Technology, China, in 2005 and his PhD degree in information science from the Poznan University of Technology, Poland, in 2014. He is currently a professor at the School of Electronic and Information Engineering in the Liaoning University of Technology, China. His research interests include combinatorial optimization, especially scheduling problems and algorithm design.



Di Wu received his BE degree in computer science from the Dalian University of Technology, China, in 2006 and his MS degree in computer application technology from that same university in 2010. His research interests include algorithm analysis and design.



Shumin Shi obtained a Bachelor's degree in statistics (big data direction) from Shangqiu Normal University in 2019. She is currently a postgraduate student majoring in computer technology at the Liaoning University of Technology, and her research interests include combinatorial optimization, scheduling problems, especially scheduling problems and algorithm design.



Liang Zhao is a lecturer at NingboTech University, a senior expert and postdoctoral researcher at Zhejiang Provincial Seaport Investment & Operation Group Co., Ltd., and a postdoctoral fellow at Ningbo University. His research interests focus on combinatorial optimization, with a specific focus on port scheduling optimization.



Jedrzej Musial is an associate professor at the Poznan University of Technology and the vice-dean for development and industrial cooperation of the Faculty of Computing and Telecommunications there. His research interests include e-commerce, Internet shopping, cloud brokering, algorithms, applications of combinatorial optimization and operations research. He holds a PhD and habilitation (DSc) in computer science from the Poznan University of Technology and a PhD from the University of Luxembourg. He has published 44 papers and two monographs.



Jacek Blazewicz is a full professor at the Poznan University of Technology (PUT) and a former director of the Institute of Computing Science, PUT. His research interests include algorithm design, computational complexity, scheduling, combinatorial optimization, bioinformatics, e-commerce. He holds a PhD and habilitation (DSc) in computer science from the PUT. His publication record includes over 400 papers in many outstanding journals. He is also the author and a co-author of over 10 monographs. His Web of Science citation index exceeds 6900 and his h-index is 39. He is an IEEE Fellow and an IFORS Fellow. More details are available on his website at `http://www.cs.put.poznan.pl/jblazewicz/`.

Received: 26 October 2025
Revised: 22 February 2026
Accepted: 11 June 2026