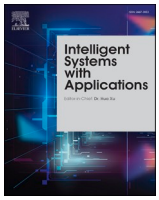
Intelligent Systems with Applications 24 (2024) 200451

Contents lists available at ScienceDirect

Intelligent Systems with Applications

journal homepage: www.journals.elsevier.com/intelligent-systems-with-applications



Reinforcement learning-based alpha-list iterated greedy for production scheduling

Kuo-Ching Ying [a], Pourya Pourhejazy [b,*], Shih-Han Cheng [c]

[a] Department of Industrial Engineering and Management, National Taipei University of Technology, Taipei 10608, Taiwan
[b] Department of Industrial Engineering, UiT- The Arctic University of Norway, Lodve Langesgate 2, Narvik 8514, Norway
[c] Microsoft Taiwan Corporation, Zhongxiao East Road, Sec. 5, Taipei City 11065, Taiwan

ARTICLE INFO

Keywords:
Production planning
Permutation flowshop
Metaheuristics
Reinforcement-learning-based algorithm
SDG 9: Industry, innovation, and infrastructure
Optimization

ABSTRACT

Metaheuristics can benefit from analyzing patterns and regularities in data to perform more effective searches in the solution space. In line with the emerging trend in the optimization literature, this study introduces the Reinforcement-learning-based Alpha-List Iterated Greedy (RAIG) algorithm to contribute to the advances in machine learning-based optimization, notably for solving combinatorial problems. RAIG uses an *N*-List mechanism for solution initialization and its solution improvement procedure is enhanced by Reinforcement Learning and an Alpha-List mechanism for more effective searches. A classic engineering optimization problem, the Permutation Flowshop Scheduling Problem (PFSP), is considered for numerical experiments to evaluate RAIG's performance. Highly competitive solutions to the classic scheduling problem are identified, with up to 9% improvement compared to the baseline, when solving large-size instances. Experimental results also show that the RAIG algorithm performs more robustly than the baseline algorithm. Statistical tests confirm that RAIG is superior and hence can be introduced as a strong benchmark for future studies.

1. Introduction

Machine learning offers solutions to many of the old and emerging engineering challenges. Engineering tools and methods are widely used to improve real-world system subprocesses through machine learning, either as a decision aid in an existing system or in determining the optimal design of a new system. Engineering optimization is a prime example of a machine learning use case with growing popularity (Park & Kwon Bae, 2015; X. Yu & Luo, 2023).

Linear and static models perform well in deterministic environments, while machine learning excels in complex and stochastic engineering optimization problems (Abaimov & Martellini, 2022). Considering the stochastic nature of metaheuristics, researchers are using machine learning to enhance the search procedure in the optimization solution space. Machine learning applications in metaheuristics can be found in optimizing production processes (Weichert et al., 2019), logistics (Giuffrida et al., 2022), geoengineering and geoscience (W. Zhang et al., 2022), hydropower operations (Bernardes et al., 2022), bioprocesses (Mondal et al., 2023), and material science (Stergiou et al., 2023), among other contexts.

Machine learning methods are typically categorized into four main types: supervised, unsupervised, semi-supervised, and reinforcement learning. In supervised learning, the model is trained on a labeled dataset, in which each input is associated with a corresponding output label. The goal is to learn a mapping from inputs to outputs, enabling the model to make predictions on unseen data. Common algorithms include decision trees, support vector machines, and neural networks. In unsupervised learning, the model is trained on data without explicit labels. The goal is to find hidden patterns or structures within the data. Common tasks include clustering, dimensionality reduction, and anomaly detection, with algorithms like k-means clustering and principal component analysis being widely used. In semi-supervised learning, the model is trained on a dataset containing a small amount of labeled data and a large amount of unlabeled data. The key idea is to leverage the abundant unlabeled data to improve learning performance; semi-supervised learning is particularly useful in scenarios where obtaining labeled data is difficult or costly while unlabeled data is readily available. Examples of semi-supervised learning algorithms include self-training, co-training, and graph-based methods. In reinforcement learning, the model learns through interacting with its environment and receiving feedback in the form of rewards or penalties. The goal is to learn a policy that maximizes the cumulative reward over time.

* Corresponding author.
E-mail addresses: kcying@ntut.edu.tw (K.-C. Ying), pourya.pourhejazy@uit.no (P. Pourhejazy), helencheng@microsoft.com (S.-H. Cheng).

https://doi.org/10.1016/j.iswa.2024.200451
Received 19 July 2024; Received in revised form 17 September 2024; Accepted 10 October 2024
Available online 11 October 2024
2667-3053/© 2024 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Reinforcement learning has applications in decision-making tasks where the outcome is influenced by a sequence of actions. Algorithms like the epsilon-greedy, Q-learning, policy gradients, and deep reinforcement learning methods are seminal examples.

Reinforcement learning-based algorithms are widely used to improve the search procedure in metaheuristics (Cheng, Pourhejazy, Ying & Lin, 2021; de Sousa Junior, Montevechi, Miranda, de Oliveira & Campos, 2020), as well as in modern engineering applications (Khosravian, Masih-Tehrani, Amirkhani & Ebrahimi-Nejad, 2024; Vakili, Amirkhani & Mashadi, 2024). Reinforcement learning in a metaheuristic algorithm involves improving the search procedure by rewarding a search trial that results in good outcomes (Rodríguez-Esparza, Masegosa, Oliva & Onieva, 2024) and, in some cases, penalizing errors and poorly performing procedures (Li, Wei, Wang, Wang & Zhang, 2024). Additionally, metaheuristics can take advantage of the reinforcement learning procedure by incorporating an automatic adjustment of search operators that considers the feedback information from prior searches (Zhang, Shao, Shao, Chen & Pi, 2024). Reinforcement learning modules have also been used to improve the efficiency of metaheuristics (Lu et al., 2024), select the best initialization method (H. Yu et al., 2024), and overcome local optima (Yue, Ma, Shi & Yang, 2024).

Integrating a learning module into the basic greedy search algorithm has resulted in notable improvements (Ying & Lin, 2022, 2023). Reinforcement learning is well-suited for the Iterated Greedy (IG) algorithm's framework for the following reasons:

(1) Adaptability to dynamic environments. Reinforcement learning can adapt to changes in the environment or problem space, making it versatile in situations where the problem's structure might evolve over time. The problem-solving process should shift as solutions are progressively refined. In this situation, adaptability becomes crucial in IGs.
(2) Exploration-Exploitation trade-off. Reinforcement learning incorporates mechanisms that balance exploration (i.e., trying new solution paths) and exploitation (i.e., refining known good solutions). This balance is important in IGs as it allows for exploring new solution paths while still focusing on improving the current solution.
(3) Learning from feedback. Reinforcement learning is designed to learn from feedback, using rewards to strengthen positive actions. In the context of the IG algorithm, this feedback-driven learning enables the algorithm to adjust its strategy based on the outcomes of previous iterations, resulting in a more effective and efficient search procedure.

These characteristics make reinforcement learning a better alternative for the IG framework compared with other machine learning techniques in terms of enhancing the performance and adaptability of the IG algorithm when solving complex optimization problems.

Among optimization approaches used for production scheduling, the IG algorithm is widely recognized due to its simplicity, flexibility, and competitive performance. However, IGs have yet to fully benefit from the use of different forms of machine learning. Among the most relevant studies, Ozsoydan and Sağir (2021) improved the IG algorithm by integrating hyper-heuristic-based learning, which acts as the intensification/diversification adaptation mechanism. Pourhejazy, Cheng, Ying and Nam (2022) integrated the meta-lamarckian learning-based perturbation mechanism into the IG algorithm to address the local optima issue.

The present study introduces the Reinforcement-learning-based Alpha-list Iterated Greedy (RAIG) algorithm to contribute to the development of machine learning-based optimization. In addition to the learning module, RAIG uses an *N*-List mechanism for solution initialization and an Alpha-List mechanism for more effective iterative neighborhood searches. This novel integration makes RAIG highly adaptable and capable of delivering robust solutions across a variety of production scheduling challenges. Extensive numerical experiments are conducted to test whether RAIG can be established as a competitive benchmark algorithm.

IG and its variations have been successfully applied to optimize complex problems like the Permutation Flowshop Scheduling Problem (PFSP). For example, Fernandez-Viagas, Ruiz and Framinan (2017) and Fernandez-Viagas and Framinan (2014) proposed the IG algorithm with local search and a tie-breaking mechanism, hereafter denoted as IG_RS$_{LS}$, which is one of the most competitive existing benchmarks for solving PFSPs. IG_RS$_{LS}$ outperformed well-known optimization algorithms such as Discrete Differential Evolution (DDE), Estimation of Distribution Algorithm (EDA), Two-Stage Bat Algorithm, Multi-Start Simulated Annealing (MSSA), Hill Climb Search, and the earlier variants of the IG and Particle Swarm Optimization (PSO) algorithms. IG_RS$_{LS}$ is, therefore, considered the baseline algorithm to evaluate the performance of RAIG when solving PFSPs. In contrast to existing approaches that primarily focus on deterministic or heuristic-based adaptations of IG, RAIG incorporates a data-driven learning component that allows the algorithm to improve continuously based on problem-specific feedback. Leveraging the strengths of reinforcement learning, our proposal enhances the performance and adaptability of IG, making it a powerful alternative to the state-of-the-art algorithms used in scheduling problems, such as IG_RS$_{LS}$.

This research article is organized into three additional sections. Section 2 introduces the proposed algorithm and elaborates on its computational components. Section 3 evaluates the developed solution algorithm by comparing it with IG_RS$_{LS}$ in solving PFSPs. Finally, Section 4 draws conclusions based on the research outcomes and suggests directions for future research.

2. Proposed method

To formalize the problem definition, let us assume that n jobs must be processed on m machines with deterministic processing times p_{ij}. In a permutation flowshop, jobs are processed in a fixed sequence on all machines, which complete the jobs in the same order. The basic permutation flowshop model assumes no interruptions once job processing begins. Additionally, machinery is the sole resource utilized in the production process. Finally, each machine can process only one task at a time and each task can be undertaken on only one machine at once. The objective is to identify the optimal job permutation that minimizes the makespan (i.e., the maximum completion time). The notations listed in Table 1 are used to clarify the solution procedure.

The pseudocode for the RAIG algorithm is presented in Fig. 1. The following subsections elaborate on the mechanisms employed in the

Table 1
Mathematical notations and symbols.

Notation	Description
i	Machine tag, where $i = 1,...,m$
j	Job index, where $j = 1,...,n$
m	Quantity of machines
n	Quantity of jobs
p_{ij}	The time it takes to process job 'i' on machine 'j'
α	Length of the candidate list
ε	A threshold to decide whether or not to retain the used α value
C_{ij}	Time of completing job 'i' on machine 'j'
L	The initial list
S	The existing partial schedule
L_N	N-list of length $1 \leq N \leq n-1$
π_S	Solutions generated by the initialization module
$\pi_{incumbent}$	The incumbent solution
π_{best}	The best solution
π_R, $\pi_{R'}$	Vector of extracted jobs and sorted extracted jobs
π_P	The job sequence after removing the extracted jobs
π_{new}	New solution in the local search phase

Procedure Epslion α-List Iterated Greedy_for_PFSP

Apply N-NEH+ heuristic to generate initial solution set $\Pi=\{\pi_s \mid s=1,...,n-1\}$
$\pi_{best} := \arg\min_{\pi_s \in \Pi}\{C_{\max}(\pi_s)\}$; $\pi_{incumbent} := \arg\min_{\pi_s \in \Pi}\{C_{\max}(\pi_s)\}$
While (termination criterion not satisfied) do
  for $k=1$ to d do // *Destruction phase*
    Remove one job at ramdom from $\pi_{incumbent}$ and add it to π_R
  endfor
  $\pi_{R'} :=$ permutation list of d removed jobs in π_R that sort in descending order of $\sum_{i=1}^{m} p_{ij}$
  $\pi_P =$ partial solution of $\pi_{incumbent}$ that after removing d jobs
  Apply epsilon-greedy stratage to select a value α ($\alpha \in \{1,...,d-1\}$) // *Construction phase*
  Use α-List NEH heuristic to sequentially reinsert all jobs of $\pi_{R'}$ into π_P until a new complete solution π_{new} is constructed
  if $C_{max}(\pi_{new}) < C_{max}(\pi_{best})$ then
    $\pi_{best} := \pi_{new}$; $\pi_{incumbent} := \pi_{new}$
    else if $C_{max}(\pi_{new}) \le C_{max}(\pi_{incumbent})$ then
      $\pi_{incumbent} := \pi_{new}$
    endif
  else if $C_{max}(\pi_{new}) > (\pi_{incumbent})$ then // Acceptance criteria
      Generate a random value r ($r \in [0,1]$)
      if $\left(r \le \exp\{-[(C_{max}(\pi_{new}) - C_{max}(\pi_{incumbent})]/C_{max}(\pi_{incumbent}))\cdot 100\}\right)$
      then $\pi_{incumbent} = \pi_{new}$
  endif
  Update fitness function value of α of the epsilon-greedy stratage
endwhile
return π_{best}

Fig. 1. Pseudocode of the RAIG algorithm.

pseudocode.

2.1. Initialization module

This study adopts the *N*-List technique developed by Puka, Duda, Stawowy and Skalna (2021) and enhances it for reinforcement learning-based applications in combinatorial optimization. The algorithm is inspired by the NEH algorithm (Nawaz, Enscore & Ham, 1983) and the concept of multiple candidate lists. The general idea involves inserting the best candidate jobs from the *N*-list (L_N) into the current partial schedule, where the unassigned jobs remain in the candidate list for the subsequent rounds and the process continues until every job is inserted into the current partial schedule, resulting in a feasible initial solution. In addition to enhancing the initial solutions, the *N*-List procedure can also be used to achieve power balance in metaheuristics. L_N has a length of $1 \le N \le n-1$ where $N=1$ reduces the algorithm to the basic form of NEH. The *N*-List mechanism provides superior diversity in the search space compared to other common initialization methods, such as random and rule-based initialization. This diversity often leads to higher-quality initial solutions, which are crucial in optimization contexts where the initial solution significantly impacts both the convergence rate and the algorithm's overall performance. Finally, the *N*-List mechanism not only improves the quality of the initial solution, but also enables the initialization module to operate effectively in parallel computing environments. This capability is advantageous over other commonly used initialization strategies and further enhances its suitability for large-scale optimization problems.

The modified procedure includes the following steps:

1. Sort the jobs in non-increasing order based on total processing time and save the sorted jobs in the initial list, $L = \{j_{[1]}, j_{[2]}, \ldots, j_{[n]}\}$.
2. Insert the first job from the initial list into the current partial schedule S; that is, $S = \{j_{[1]}\}$, and remove it from L.
3. Set the candidate list as $L_N = \{j_{[2]}, \ldots, j_{[N+1]}\}$. Evaluate each job in L_N by placing it in all possible positions within the existing partial schedule, S. Select the job that yields the best outcome and remove this job from L_N. Set the best outcome as the current partial schedule.
4. If $L \neq \phi$, move the first job from L to L_N and remove it from L.
5. Repeat steps 3 and 4 until all jobs are inserted into S.

This procedure is illustrated with an example. Consider an example based on the information shown in Table 2. Fig. 2 illustrates this procedure for the case when $L_N = 1$. After extracting the first job and placing it in every possible position, the best outcome is considered the best partial schedule; this procedure continues until all jobs are inserted.

The initialization procedure differs when $L_N > 1$. For $L_N = 2$, every time a job needs to be inserted, two candidates from the *N*-List are considered to explore all possibilities. The illustrative procedure is depicted in Fig. 3. In this example, the jobs are initially sorted in descending order based on total processing time. The job with the largest total processing time, i.e., job 3, is inserted first. In the next step, the two candidates from L_N are selected, and inserted into every possible position of the partial schedule to identify the most-competitive alternative.

Table 2
The illustrative example.

Job	M_1	M_2	M_3	M_4	M_5	Total processing time
J_1	7	59	22	73	38	199
J_2	92	33	73	22	54	274
J_3	75	66	32	64	42	279
J_4	44	5	53	51	20	173
J_5	25	15	10	24	21	95

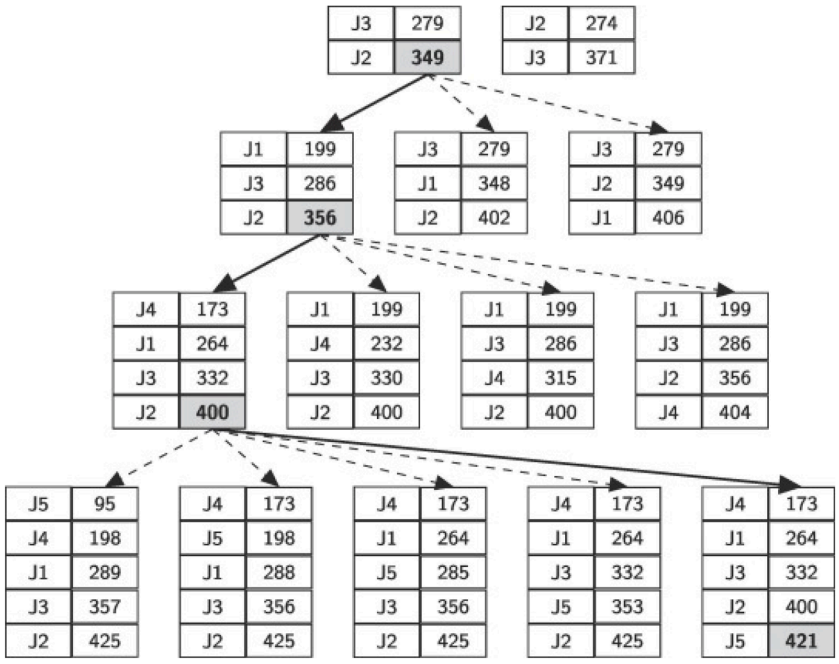


Fig. 2. Visual illustration of solution initialization when $L_N = 1$.

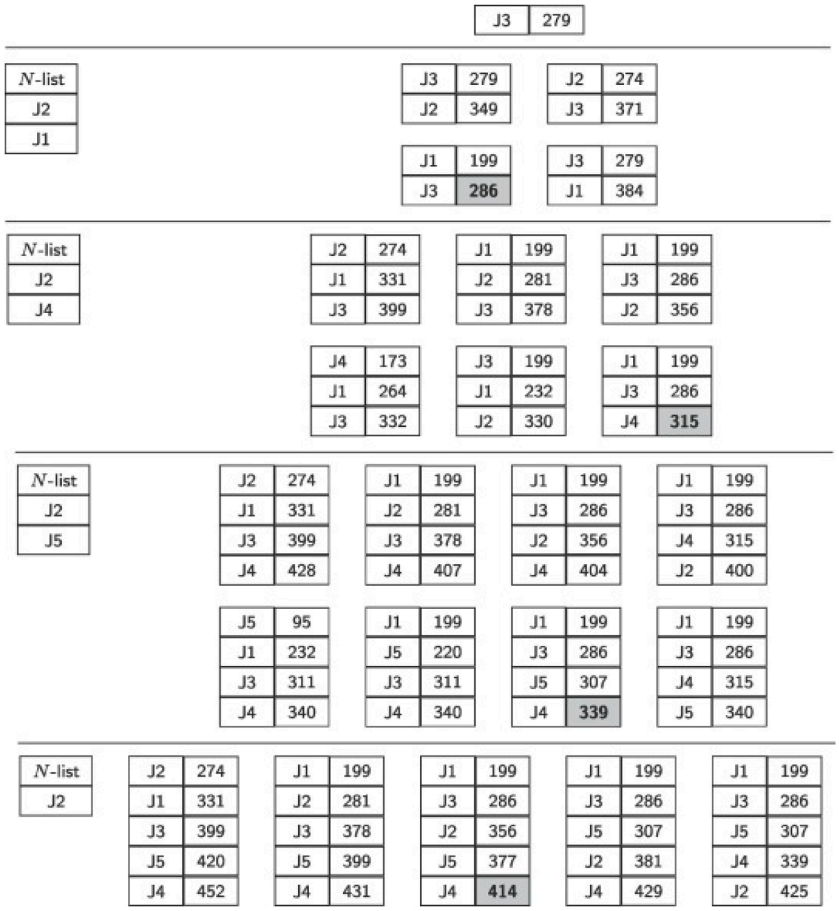


Fig. 3. Visual illustration of solution initialization when $L_N = 2$.

This procedure results in the partial schedule $\{j_1, j_3\}$ in the second step, $\{j_1, j_3, j_4\}$ in the third step, $\{j_1, j_3, j_5, j_4\}$ in the fourth step, and the complete schedule of $\{j_1, j_3, j_2.j_5, j_4\}$. It is worth noting that If $L_N = 3$, three candidates will be considered for insertion. Comparison of the outcomes in Figs. 2 and 3 shows that $L_N = 2$ results in a better initial solution. This difference may become more significant when solving

industry-scale problems.

2.2. Epsilon-greedy mechanism

The IG algorithm has demonstrated a strong exploitation power, but it is prone to falling into local optima traps. The imbalance between the exploitation and exploration abilities often results in suboptimal situations. To address this issue, the epsilon-greedy policy was developed, which iteratively uses reinforcement learning to reconcile the solution algorithm's exploration and exploitation in stochastic search environments. The learning module helps the search algorithm prioritize actions with the best outcomes through a rewarding mechanism. The reward function, shown in Eq. (1), is used as the basis of the rewarding mechanism, where V_t represents the value of the selected action, i.e., α; n indicates the total number of times the action has been selected; V_{t-1} is the value calculated for the last selected action; R_t is the reward value assigned to the selected action.

The epsilon-greedy selection strategy is a straightforward yet powerful approach commonly employed in reinforcement learning algorithms to balance exploration and exploitation. This method regulates the balance by incorporating a parameter known as "epsilon" (ε), which defines the probability of engaging in exploration. At each decision-making step, the algorithm samples a random number from a uniform distribution $U(0,1)$. If this number is below the epsilon (ε) threshold, the algorithm selects an action randomly, leveraging the roulette wheel selection method to facilitate exploration. Otherwise, if the number meets or exceeds ε, the algorithm chooses the action with the highest fitness value based on prior steps, favoring exploitation. Overall, higher values of epsilon favor exploration, while lower values favor exploitation.

Inspired by the epsilon-greedy selection strategy, the fitness function in Eq. (2) regulates the rewards considering the makespan value of the job schedules, where fit^A_{iter} represents the outcome (fitness) of the selected action to be used in the upcoming round of roulette wheel selection; n^A is the total number of times action A has been selected in the solution procedure. If the obtained makespan value is better in this trial, the updated fitness function value of the selected action will be more conducive to the next choice; otherwise, a lower fitness function value decreases the odds of selecting the action in the upcoming iteration.

$$V_t = \frac{n-1}{n} \times V_{t-1} + \frac{1}{n} \times R_t \tag{1}$$

$$fit^N_{iter} = \frac{n^N - 1}{n^N} \times fit^N_{iter-1} + \frac{1}{n^N} \times [C_{\max}(\pi) - C_{\max}(\pi')] \tag{2}$$

2.3. Destruction/construction mechanism

Jobs are drawn randomly to re-construct an existing solution. The destruction parameter, d, indicates how many jobs are drawn in the destruction procedure, and the most competitive setting is selected through calibration experiments. The jobs extracted during the destruction phase are saved in vector π_R.

Next, the α-List is used to arrange the extracted jobs in order of the total processing time to carry out the construction procedure. In each construction cycle, the jobs listed in the α-List are inserted into every possible position of the current partial schedule, one by one, and the option with the best makespan value is selected. The acceptance mechanism decides whether to update the current best and incumbent solutions.

2.4. Acceleration method

NEH (Nawaz et al., 1983) is extensively used as a constructive heuristic for solving PFSP and its variants. The NEH process consists of: (1) calculating the overall processing times of jobs and sorting them in non-decreasing order; (2) scheduling the first two jobs from the initial sequence and inserting the next jobs into the partial solution where the makespan is the least (a total of k possibilities), which have the computational complexities of $O(mn + n\log n)$ and $O(mn^3)$, respectively. Thus, the overall time complexity is $O(mn^3)$. In this study, the acceleration approach proposed by Taillard (1993) is applied to reduce the RAIG algorithm's computational complexity.

2.5. Acceptance mechanism

The acceptance mechanism developed by Ruiz, Pan and Naderi (2019) is adopted in this study. This approach is inspired by the annealing processes of metals and uses Eq. (3) to determine the acceptance or rejection of a new, worse solution. Given that the existing solution is π and the 'worse' new solution under acceptance consideration is π', the acceptance probability is calculated based on the current temperature (T) and the relative difference between their respective fitness values, i.e., $(C_{max}(\pi') - C_{max}(\pi))/C_{max}(\pi)$. This approach uses a random value that does not exceed the acceptance probability as a criterion to accept a worse new solution; otherwise, the search continues with the current schedule until the termination condition is met. The temperature value is derived from Eq. (4), where T refers to the initial temperature and is adjusted after every calculation. Assuming a fixed temperature value, a smaller difference between makespan values increases the acceptance probability. Alternatively, a greater difference between makespan values tightens the acceptance threshold. The acceptance mechanism is less restrictive at the beginning of the search and gradually becomes more restrictive as the search progresses. Finally, considering the relative percentage difference (*RPD*; Eq. (5)) allows us to differentiate situations with similar makespan value differences (e.g., 100 & 110 vs. 1000 & 1010, where the difference in the former is more significant).

$$Random \leq e^{\frac{-RPD}{Temperature}} \tag{3}$$

$$Temperature = T \times \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} P_{ij}}{n \times m \times 10} \tag{4}$$

$$RPD = \frac{C_{\max}(\pi') - C_{\max}(\pi)}{C_{\max}(\pi)} \times 100 \tag{5}$$

The initialization module, destruction/construction mechanism, and acceptance mechanism with the acceleration method have computational complexities of $O(mn^3)$, $O(n^3)$, and $O(n)$, respectively. Thus, the overall time complexity of RAIG is $O(mn^3)$, which is the same as that of the original IG and the IG_RS$_{LS}$ algorithm.

3. Numerical analysis

3.1. Preliminaries

The IG_RS$_{LS}$ algorithm (Fernandez-Viagas & Framinan, 2014), which is one of the best-performing algorithms for solving PFSPs, is considered to be the baseline algorithm for evaluating RAIG's performance. The development environment for all compared algorithms was Microsoft Visual Studio 2022 with C++ being the programming language. Code compilation was performed with the Microsoft Visual C++ compiler, applying the /O2 optimization flag to maximize execution speed by enabling high-level optimizations like loop unrolling, function inlining, and advanced memory usage optimizations. On the hardware side, the experiments were conducted on a personal computer equipped with an Intel$^{(R)}$ Core$^{(TM)}$ i7–7700 CPU running at a base clock frequency of 3.60 GHz. This processor features 4 physical cores and 8 threads, allowing for efficient multi-threaded execution. The machine was also configured with 16 GB of DDR4 RAM, providing ample memory for the algorithms'

computational requirements. The operating system was Windows 10 (64-bit), ensuring smooth execution and access to necessary system resources for the compiled binaries.

This study uses the standard dataset provided by Taillard (1993) to investigate the algorithms' performance. The dataset consists of 120 instances with 12 different configurations. The configurations are characterized by different workloads (i.e., various job quantities) with $n \in \{20, 50, 100, 200, 500\}$, different number of machines with $m \in \{5, 10, 20\}$, and job processing times generated using a *Uniform* $[1, 99]$ distribution. Each instance is solved five times using each of the solution algorithms (a total of 600 experiments), and the best, worst, and average fitness values are recorded for comparative analysis.

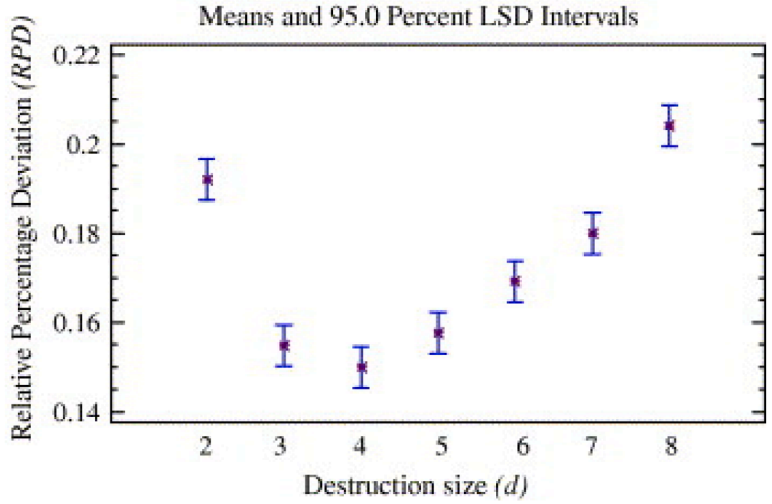
Finally, the algorithms are compared in terms of overall performance for various workloads and the number of machines on the shop floor. To evaluate performance, the best, worst, and average makespan values obtained by each algorithm are used to calculate the *RPD* using Eq. (7). In this formulation, $C_{max}(\pi)$ denotes the makespan of the solution under evaluation, while $C_{max}(\pi_{best})$ refers to the best-found makespan for the particular instance. An *RPD* value of 0 represents the best solution, and a smaller *RPD* value indicates a more competitive solution obtained by the algorithm. Since the average value and comparisons between groups are considered for performance evaluation, the results are presented in the form of Average Relative Percentage Deviation (ARPD).

$$RPD = \frac{\text{Cmax}(\pi) - \text{Cmax}(\pi_{best})}{\text{Cmax}(\pi_{best})} \times 100 \tag{7}$$

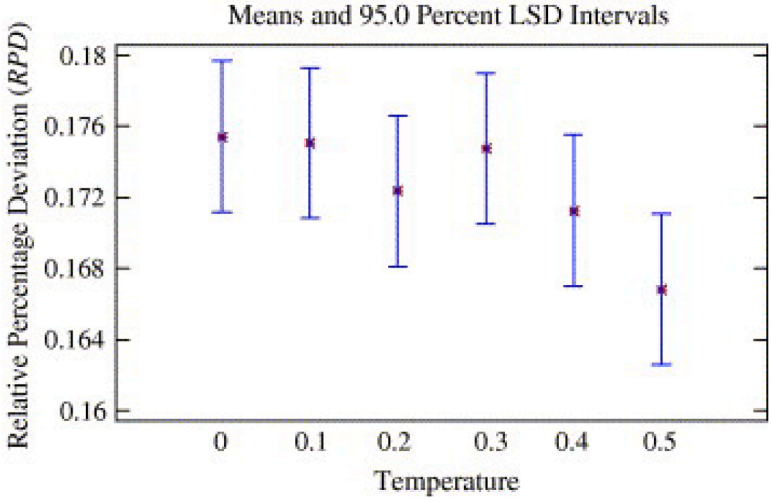
3.2. Parameter calibration

To select the best parameter settings, different levels for the main parameters of the RAIG algorithm are evaluated: d and T and ε; d denotes the destruction size; T refers to the initial temperature of the acceptance mechanism; and ε stands for a threshold to decide whether or not to retain the value of α used in the current iteration.

Since the algorithm is an extension of the IG algorithm, the parameter setting was primarily based on the work of Ruiz and Stützle (2007). For the calibration experiments, 12 test sets, each consisting of five test instances, were randomly generated using the same configurations as those of test instances provided by Taillard (1993). Following Ruiz and Stützle (2007), Fisher's Least Significant Difference (LSD) with 95 percent confidence was used for statistical analysis. As shown in Fig. 4 (a), the calibration results showed that the RAIG algorithm achieved the



(a) Destruction count



(b) Temperature

Fig. 4. Calibration reference.

best *RPD* values when the destruction size was set to $d = 4$. The calibration results in Fig. 4(b) confirmed that $T = 0.5$ was the best initial temperature value to proceed with the calibration experiments.

According to Dos Santos Mignon and De Azevedo da Rocha (2017), when the value of ε exceeds 0.5, there is no significant gain in performance. Therefore, in this study, ε within the range of 0.1 to 0.4 was considered for calibration. Each of the 12 test instances for calibration experiments was solved using the RAIG algorithm with different values of ε, and the best value of ε for each set was summarized in Table 3. In addition, following Fernandez-Viagas and Framinan (2014), the maximum computational time was set as the stopping condition for a fair evaluation of the algorithms. To achieve a balance between computational efficiency and solution quality, $n \times (m/2) \times t$ was used to define the maximum computational time, considering the workload, the number of machines, and $t = \{30, 60, 90\}$, where $t = 90$ yielded the best outcomes for solving PFSPs (Fernandez-Viagas & Framinan, 2014).

Based on the range of *ARPDs*, the parameter d is the most sensitive one of all parameters. Under a fixed maximum computation time, a higher value of d would result in more jobs being removed from the incumbent solution, thereby increasing the computation time allocated to the subsequent construction mechanism for each algorithmic iteration, which in turn leads to a reduction in the total number of iterations. Hence, an excessively large value of d would lead to a reduced number of candidate solution evaluations, potentially hindering the discovery of the best solution. The second most sensitive parameter is T, which directly affects the acceptance probability of a new, worse solution in the greedy search. A larger initial temperature value of the acceptance mechanism tightens the acceptance threshold; thus, a suitable value of T could balance the exploration-exploitation dilemma of the RAIG algorithm. In contrast, parameter ε exhibits relatively minor sensitivity to the performance of RAIG. No significant differences were observed between the *ARPDs* corresponding to different ε values within the range of 0.1 to 0.4.

3.3. Results analysis

Considering a total of 120 test instances, the number of optimal solutions found by the algorithms is first analyzed. This is followed by the number of times the RAIG algorithm outperforms the baseline algorithm in terms of the best (Min), average (Avg), and worst (Max) fitness values. It can be observed in Table 4 that RAIG performed better by yielding the optimal solutions in more instances compared with IG_RS$_{LS}$. Expectedly, neither algorithm was able to converge to the optimal solution when solving very large instances.

The small difference between the worst (79), best (86), and average performance (79) of RAIG shows that the algorithm's performance is quite stable. Yielding 79 out of the 120 best-found solutions demonstrates improved stability relative to the state-of-the-art IG.

Table 5 compares the algorithms using the *RPD* and *ARPD* metrics for each instance group. RAIG outperforms the baseline algorithm in 27 out of 36 cases and also yields smaller *ARPDs* for all three metrics.

Table 3
Calibration results for RAIG.

Calibration test set	Corresponding test instances	Best value of ε
TN001	Ta001-Ta010	0.2
TN002	Ta011-Ta020	0.3
TN003	Ta021-Ta030	0.3
TN004	Ta031-Ta040	0.1
TN005	Ta041-Ta050	0.1
TN006	Ta051-Ta060	0.3
TN007	Ta061-Ta070	0.4
TN008	Ta071-Ta080	0.3
TN009	Ta081-Ta090	0.4
TN010	Ta091-Ta100	0.1
TN011	Ta101-Ta110	0.1
TN012	Ta111-Ta120	0.4

Table 4
Overall performance in terms of optimality counts.

Workload (total number of instances)	Optimality counts	
	IG_RS$_{LS}$	RAIG
20(30)	10	13
50(30)	5	6
100(30)	5	8
200(20)	0	0
500(10)	0	0

	RAIG (Min)	RAIG (Avg)	RAIG(Max)
Best Found Solutions	49	72	63
Tie	30	7	23

Table 5
Overall performance in terms of relative percentage deviation (best in **bold**).

Instances	Min		Avg		Max	
	IG_RS$_{LS}$	RAIG	IG_RS$_{LS}$	RAIG	IG_RS$_{LS}$	RAIG
001:010	**0.1842**	**0.1842**	0.4923	**0.3452**	0.9688	**0.5360**
011:020	0.3431	**0.2669**	0.8249	**0.6317**	1.3198	**0.9580**
021:030	0.2834	**0.2312**	0.6426	**0.5334**	1.0455	**0.8759**
031:040	0.0964	**0.0889**	**0.1759**	0.1863	**0.2573**	0.3057
041:050	**1.2134**	1.2715	1.7413	**1.6397**	2.2898	**2.1260**
051:060	2.1175	**2.0842**	2.7490	**2.5794**	3.3191	**3.1073**
061:070	0.0439	**0.0193**	0.0836	**0.0778**	**0.1722**	0.1753
071:080	0.6372	**0.4936**	**0.7581**	0.7785	**0.8917**	0.9629
081:090	2.1468	**2.1268**	**2.4436**	2.5172	**2.8455**	2.9192
091:100	0.3516	**0.3237**	0.5611	**0.4648**	0.7718	**0.6171**
101:110	**1.7636**	1.8673	2.2580	**2.2881**	2.7774	**2.5787**
111:120	0.9223	**0.8742**	1.0491	**0.9901**	1.1909	**1.0900**
Average	0.8420	**0.8193**	1.1483	**1.0860**	1.4875	**1.3543**

Smaller ARPDs mean that integrating the learning modules resulted in more effectiveness in the search procedure. The difference between ARPD values becomes greater when considering the algorithms' best performance. The performance of the algorithms is further analyzed by considering various problem sizes. Tables 6-7 summarize the results for various workloads with and without considering the number of machines, respectively.

It is observed that RAIG performs better in terms of all the *ARPD* values and all problem categories. The difference becomes wider in the instances with the largest workload (i.e., 500 jobs) and larger workshops (i.e., 20 machines).

As the final step in the numerical experiments, the paired sample *t*-tests are conducted to determine if the *RPDs* obtained from the benchmarks are meaningfully different. For this purpose, a one-tailed *t*-test is used to determine whether RAIG performs significantly better than the compared algorithm. The minimum, average, and maximum values are considered separately for the tests; the results are summarized in Table 8.

The statistical results, with a 95 percent confidence level, provide dependable evidence that RAIG outperforms the baseline algorithm considering the average and maximum *RPDs*. This confirms that RAIG offers more reliable performance across a broad range of cases. The improvement in the best fitness values when solving the small-scale problems are not significant; the small *p-value* in the full range row, however, implies that, while RAIG may not always significantly improve upon the best outcome in every instance, it consistently produces better-than-average results, making it a reliable algorithm in general.

Overall, one can claim that RAIG performs significantly more effectively and robustly than the baseline algorithm. In terms of scalability, the largest instances tested in the present study include 500 jobs and 20 machines, and the results indicated that RAIG maintains its competitiveness as the problem size increases. However, further analysis is required to assess RAIG's computational efficiency in handling larger and more complex instances. This will provide a basis for adapting RAIG

Table 6
Relative performance deviation by the number of machines (best in **bold**).

Machines	Jobs	Min		Avg		Max	
		IG_RS_{LS}	RAIG	IG_RS_{LS}	RAIG	IG_RS_{LS}	RAIG
5	20	0.1842	0.1842	0.4923	0.3452	0.9688	0.5360
	50	0.0964	0.0889	0.1759	0.1863	0.2573	0.3057
	100	0.0439	0.0193	0.0836	0.0778	0.1722	0.1753
	Average	0.1082	**0.0975**	0.2506	**0.2031**	0.4661	**0.3390**
10	20	0.3431	0.2669	0.8249	0.6317	1.3198	0.9580
	50	1.2134	1.2715	1.7413	1.6397	2.2898	2.1260
	100	0.6372	0.4936	0.7581	0.7785	0.8917	0.9629
	200	0.3516	0.3237	0.5611	0.4648	0.7718	0.6171
	Average	0.6363	**0.5889**	0.9713	**0.8787**	1.3183	**1.1660**
20	20	0.2834	0.2312	0.6426	0.5334	1.0455	0.8759
	50	2.1175	2.0842	2.7490	2.5794	3.3191	3.1073
	100	2.1468	2.1268	2.4436	2.5172	2.8455	2.9192
	200	1.7636	1.8673	2.2580	2.2881	2.7774	2.5787
	500	0.9223	0.8742	1.0491	0.9901	1.1909	1.0900
	Average	1.4467	**1.4367**	1.8285	**1.7816**	2.2357	**2.1142**

Table 7
Average relative performance deviation considering the number of jobs.

Workload	Min		Avg		Max	
	IG_RS_{LS}	RAIG	IG_RS_{LS}	RAIG	IG_RS_{LS}	RAIG
20	0.2702	0.2274	0.6533	0.5034	1.1114	0.7900
50	1.1424	1.1482	1.5554	1.4685	1.9554	1.8463
100	1.3920	1.3102	1.6008	1.6479	1.8686	1.9411
200	1.0576	1.0955	1.4095	1.3765	1.7746	1.5979
500	0.9223	0.8742	1.0491	0.9901	1.1909	1.0900
ARPD	0.9569	**0.9311**	1.2536	**1.1973**	1.5802	**1.4531**

Table 8
Statistical results for the performance evaluation of the algorithms.

	Degree of Freedom	*T*	*p*-value
Minimum	119	0.812	0.209
Average	119	1.975	0.025
Maximum	119	3.020	0.001
Full range	119	2.547	0.006

as an optimization tool for developing decision-aid platforms.

4. Conclusions

Machine learning enables metaheuristic algorithms to perform better throughout the search procedure by learning from the identified search patterns and structures; this facilitates a more effective and efficient move toward global optima. This study introduces the RAIG algorithm, which is equipped with reinforcement learning and the α-List mechanism for improved greedy search, as well as an *N*-List mechanism for solution initialization. The reinforcement learning-based construction mechanism uses a candidates list of length α for job insertions, where the ε-greedy module decides whether to update the list.

As a classic combinatorial optimization problem, PFSP is used to evaluate the algorithm's performance by comparing it with a competitive variant of the IG algorithm. The numerical analysis confirmed a meaningful difference between the average and maximum makespans found by the RAIG and baseline algorithms. Overall, the performance of RAIG is about 6–9 percent better when solving large instances, while the overall performance improvement is about 3 percent. It is also shown that RAIG is significantly more robust than the baseline algorithm.

The classic optimization problem considered in this study is a fairly simple variant of flowshop scheduling. This represents a limitation that necessitates further research to evaluate the performance of RAIG in comparison with the best-performing algorithms for more complex combinatorial optimization problems. This may involve considering other performance indicators and incorporating multi-objective optimization to solve production scheduling problems with additional constraints. Conducting an empirical analysis comparing the proposed RAIG algorithm with a wider range of state-of-the-art deep learning or reinforcement learning-based methods in terms of accuracy, efficiency, computational complexity, and inference speed merits further investigation. Validating these algorithms in real-world industrial settings, which requires collaboration with industry partners and access to real-world data, is another important and valuable area for future research. From a search algorithm perspective, considering multiple initial solutions instead of a single solution could improve the search procedure and address the issue of getting trapped in local optima. Additionally, proposing concrete frameworks for integrating lower-bound identification modules could provide clear pathways for further advancements. For instance, integrating a matheuristic-inspired algorithm with the learning module may further enhance search effectiveness. Last but not least, exploring RAIG's applications in other combinatorial optimization problems could highlight its versatility and potential impact.

CRediT authorship contribution statement

Kuo-Ching Ying: Conceptualization, Methodology, Software, Supervision, Writing – review & editing. **Pourya Pourhejazy:** Investigation, Writing – original draft. **Shih-Han Cheng:** Formal analysis, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Abaimov, S., & Martellini, M. (2022). Understanding Machine Learning. In *Machine Learning for Cyber Agents. Advanced Sciences and Technologies for Security Applications*. Cham: Springer. https://doi.org/10.1007/978-3-030-91585-8_2.

Bernardes, J., Santos, M., Abreu, T., Prado, L., Miranda, D., Julio, R., et al. (2022). Hydropower operation optimization using machine learning: A systematic review. *AI, 3*(1), 78–99. https://doi.org/10.3390/ai3010006

Burcin Ozsoydan, F., & Sağir, M. (2021). Iterated greedy algorithms enhanced by hyper-heuristic based learning for hybrid flexible flowshop scheduling problem with sequence dependent setup times: A case study at a manufacturing plant. *Computers &*

Operations Research, 125, Article 105044. https://doi.org/10.1016/J.COR.2020.105044

Cheng, C.-Y., Pourhejazy, P., Ying, K.-C., & Lin, C.-F. (2021). Unsupervised learning-based artificial bee colony for minimizing non-value-adding operations. *Applied Soft Computing, 105*, Article 107280. https://doi.org/10.1016/j.asoc.2021.107280

de Sousa Junior, W. T., Montevechi, J. A. B., Miranda, R.de C., de Oliveira, M. L. M., & Campos, A. T. (2020). Shop floor simulation optimization using machine learning to improve parallel metaheuristics. *Expert Systems with Applications, 150*, Article 113272. https://doi.org/10.1016/J.ESWA.2020.113272

Dos Santos Mignon, A., & De Azevedo Da Rocha, R. L. (2017). An adaptive implementation of ε-greedy in reinforcement learning. *Procedia Computer Science, 109*, 1146–1151. https://doi.org/10.1016/J.PROCS.2017.05.431

Fernandez-Viagas, V., & Framinan, J. M. (2014). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research, 45*, 60–67. https://doi.org/10.1016/J.COR.2013.12.012

Fernandez-Viagas, V., Ruiz, R., & Framinan, J. M. (2017). A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research, 257*(3), 707–721. https://doi.org/10.1016/j.ejor.2016.09.055

Giuffrida, N., Fajardo-Calderin, J., Masegosa, A. D., Werner, F., Steudter, M., & Pilla, F. (2022). Optimization and machine learning applied to last-mile logistics: A review. *Sustainability, 14*(9), 5329. https://doi.org/10.3390/su14095329

Khosravian, A., Masih-Tehrani, M., Amirkhani, A., & Ebrahimi-Nejad, S. (2024). Robust autonomous vehicle control by leveraging multi-stage MPC and quantized CNN in HIL Framework. *Applied Soft Computing, 162*, Article 111802. https://doi.org/10.1016/j.asoc.2024.111802

Li, C., Wei, X., Wang, J., Wang, S., & Zhang, S. (2024). A review of reinforcement learning based hyper-heuristics. *PeerJ Computer Science, 10*, e2141. https://doi.org/10.7717/peerj-cs.2141

Lu, R., Jiang, Z., Yang, T., Chen, Y., Wang, D., & Peng, X. (2024). A novel hybrid-action-based deep reinforcement learning for industrial energy management. *IEEE Transactions on Industrial Informatics*, 1–15. https://doi.org/10.1109/TII.2024.3424529

Mondal, P. P., Galodha, A., Verma, V. K., Singh, V., Show, P. L., Awasthi, M. K., et al. (2023). Review on machine learning-based bioprocess optimization, monitoring, and control systems. *Bioresource Technology, 370*, Article 128523. https://doi.org/10.1016/J.BIORTECH.2022.128523

Nawaz, M., Enscore, E. E., Jr, & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*(1), 91–95.

Park, B., & Kwon Bae, J. (2015). Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data. *Expert Systems with Applications, 42*(6), 2928–2934. https://doi.org/10.1016/J.ESWA.2014.11.040

Pourhejazy, P., Cheng, C.-Y., Ying, K.-C., & Nam, N. H. (2022). Meta-Lamarckian-based iterated greedy for optimizing distributed two-stage assembly flowshops with mixed setups. *Annals of Operations Research*. https://doi.org/10.1007/s10479-022-04537-2

Puka, R., Duda, J., Stawowy, A., & Skalna, I. (2021). N-NEH+ algorithm for solving permutation flow shop problems. *Computers & Operations Research, 132*, Article 105296. https://doi.org/10.1016/j.cor.2021.105296

Rodríguez-Esparza, E., Masegosa, A. D., Oliva, D., & Onieva, E. (2024). A new hyper-heuristic based on adaptive simulated annealing and reinforcement learning for the capacitated electric vehicle routing problem. *Expert Systems with Applications, 252*, Article 124197. https://doi.org/10.1016/j.eswa.2024.124197

Ruiz, R., Pan, Q.-K., & Naderi, B. (2019). Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega, 83*, 213–222. https://doi.org/10.1016/j.omega.2018.03.004

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049. https://doi.org/10.1016/j.ejor.2005.12.009

Stergiou, K., Ntakolia, C., Varytis, P., Koumoulos, E., Karlsson, P., & Moustakidis, S. (2023). Enhancing property prediction and process optimization in building materials through machine learning: A review. *Computational Materials Science, 220*, Article 112031. https://doi.org/10.1016/J.COMMATSCI.2023.112031

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*(2), 278–285.

Vakili, E., Amirkhani, A., & Mashadi, B. (2024). DQN-based ethical decision-making for self-driving cars in unavoidable crashes: An applied ethical knob. *Expert Systems with Applications, 255*, Article 124569. https://doi.org/10.1016/j.eswa.2024.124569

Weichert, D., Link, P., Stoll, A., Rüping, S., Ihlenfeldt, S., & Wrobel, S. (2019). A review of machine learning for the optimization of production processes. *The International Journal of Advanced Manufacturing Technology, 104*(5–8), 1889–1902. https://doi.org/10.1007/s00170-019-03988-5

Ying, K.-C., & Lin, S.-W. (2022). Reinforcement learning iterated greedy algorithm for distributed assembly permutation flowshop scheduling problems. *Journal of Ambient Intelligence and Humanized Computing*. https://doi.org/10.1007/s12652-022-04392-w

Ying, K.-C., & Lin, S.-W. (2023). Minimizing makespan in two-stage assembly additive manufacturing: A reinforcement learning iterated greedy algorithm. *Applied Soft Computing, 138*, Article 110190. https://doi.org/10.1016/J.ASOC.2023.110190

Yu, H., Gao, K., Wu, N., Zhou, M., Suganthan, P. N., & Wang, S. (2024). Scheduling multiobjective dynamic surgery problems via *Q* -learning-based meta-heuristics. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 54*(6), 3321–3333. https://doi.org/10.1109/TSMC.2024.3352522

Yu, X., & Luo, W. (2023). Reinforcement learning-based multi-strategy cuckoo search algorithm for 3D UAV path planning. *Expert Systems with Applications, 223*, Article 119910. https://doi.org/10.1016/J.ESWA.2023.119910

Yue, B., Ma, J., Shi, J., & Yang, J. (2024). A deep reinforcement learning-based adaptive search for solving time-dependent green vehicle routing problem. *IEEE access : practical innovations, open solutions, 12*, 33400–33419. https://doi.org/10.1109/ACCESS.2024.3369474

Zhang, W., Gu, X., Tang, L., Yin, Y., Liu, D., & Zhang, Y. (2022). Application of machine learning, deep learning and optimization algorithms in geoengineering and geoscience: Comprehensive review and future challenge. *Gondwana Research, 109*, 1–17. https://doi.org/10.1016/J.GR.2022.03.015

Zhang, Z., Shao, Z., Shao, W., Chen, J., & Pi, D. (2024). MRLM: A meta-reinforcement learning-based metaheuristic for hybrid flow-shop scheduling problem with learning and forgetting effects. *Swarm and Evolutionary Computation, 85*, Article 101479. https://doi.org/10.1016/j.swevo.2024.101479