**An iterated greedy algorithm with memory and learning mechanisms for the distributed permutation flow shop scheduling problem**

**--Supplementary Materials**

This document provides the supplementary material for the article entitled “An iterated greedy algorithm with memory and learning mechanisms for the distributed permutation flow shop scheduling problem”, which has been submitted to the special issue of Computers, Materials & Continua, i.e., Recent Advances in Ensemble Framework of Meta-heuristics and Machine Learning: Methods and Applications.

**1 Literature review**

Table S1 gathers some outstanding literature of solving DPFSPs from the last decade, which is categorized according to the algorithm, learning, decision, local search and acceptance criterion. We only label acceptance criteria of the iterated greedy algorithm (IGA)-related literature. With respect to learning and decision, we only label the literature that uses reinforcement learning (RL) mechanisms. For ease of differentiation, IGA-related literature has been shaded in gray. Among these literature in Table S1, only Feng, Zhao, Jiang, Tao and Mei [1] considers the memory mechanism. Therefore, the information about the memory mechanism is not reflected in Table S1.

Table S1:Classification of some outstanding literature from the last decade

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Reference | Algorithm | Learning | Decision | Local search | Acceptance |
| Naderi and Ruiz (2010) [[2]](#bookmark4) | VNS |  |  | Insr |  |
| Naderi and Ruiz (2014) [[3]](#bookmark10) | SS |  |  | Insr |  |
| Wang and Wang (2015) [[4]](#bookmark11) | EDA, MA |  |  | Insr, Swap, Revr |  |
| Fernandez-Viagas and Framinan (2015) [[5]](#bookmark12) | IGA |  |  | Insr | SA |
| Lin and Zhang (2016) [[6]](#bookmark13) | BBO |  |  | Insr |  |
| Lin and Ying (2016) [[7]](#bookmark14) | ICG |  |  | Insr |  |
| Bargaoui, Driss and Ghdira (2017) [[8]](#bookmark15) | CRO |  |  | 1p-CO |  |
| Deng and Wang (2017) [[9]](#bookmark16) | MA |  |  | Insr, Swap |  |
| Lin, Wang and Li (2017) [[10]](#bookmark17) | BSA, HH |  |  | Insr, Swap |  |
| Shao, Pi and Shao (2017) [[11]](#bookmark18) | IGA |  |  | Insr, Swap | SA |
| Fernandez-Viagas, Perez-Gonzales and Framinan (2018) [[12]](#bookmark19) | EA |  |  | Insr |  |
| Pan, Gao, Li and Framinan (2019) [[13]](#bookmark20) | IGA |  |  | Insr | SA |
| Ruiz, Pan and Naderi (2019) [[14]](#bookmark5) | IGA |  |  | Insr | SA |
| Wang and Wang (2019) [[15]](#bookmark21) | IGA |  |  | Insr, Swap | Greedy |
| Meng, Pan and Wang (2019) [[16]](#bookmark22) | IGA, VND, ABC |  |  | Insr, Swap | SA |
| Shao, Shao and Pi (2020) [[17]](#bookmark23) | IGA |  |  | Insr, Swap | SA |
| Pan, Gao and Wang (2020) [[18]](#bookmark24) | EA |  |  | Shift, Swap |  |
| Huang, Pan and Gao (2020) [[19]](#bookmark25) | IGA |  |  | B-Insr | Greedy, Restart |
| Zhao, Ma and Wang (2021) [[20]](#bookmark26) | Jaya |  |  | Insr, Swap |  |
| Li, Pan, Li, Gao and Tasgetiren (2021) [[21]](#bookmark27) | IGA |  |  | Insr, Swap | Greedy, Restart |
| Shao, Shao and Pi (2021) [[22]](#bookmark28) | IGA |  |  | Insr, Swap | SA |
| Mao, Pan, Miao and Gao (2021) [[23]](#bookmark29) | IGA |  |  | Insr, Swap | SA |
| Zhang, Qian, Hu, Jin and Wang (2021) [[24]](#bookmark30) | EDA |  |  | Insr, Swap, Revr |  |
| Li, Li and Gao (2021) [[25]](#bookmark31) | ABC |  |  | Insr, Swap |  |
| Shao, Shao and Pi (2021) [[26]](#bookmark32) | EA |  |  | Insr, Swap |  |
| Li, Pan, Gao, Tasgetiren, Zhang and Li (2021) [[27]](#bookmark33) | GA |  |  | Insr, Swap |  |
| Chen, Pan, Gao and Sang (2021) [[28]](#bookmark34) | IGA |  |  | Insr, Swap | Greedy |
| Huang, Pan, Huang, Suganthan and Gao (2021) [[29]](#bookmark35) | IGA |  |  | Insr | Greedy |
| Zhao, Zhao, Wang and Tang (2021) [[30]](#bookmark36) Zhao, Zhang, Cao and Tang (2021) [[31]](#bookmark37) Yang and Li (2022) [[32]](#bookmark38) | BSAWWO KDH | Q-learning\*1 | Opt | Insr, Swap, B-Shift Insr, Swap, DCInsr |  |
| Li, Pan, He, Sang, Gao and Jing (2022) [[33]](#bookmark39) | IGA |  |  | Insr | SA |
| Shao, Shao and Pi (2022) [[34]](#bookmark40) | NMA |  |  | Insr, Swap |  |
| Hou, Fu, Gao, Zhang and Sadollan (2022) [[35]](#bookmark41) | BSO |  |  | Insr, Swap, CO |  |
| Zhang, Liu, Wang, Yu and Xing (2022) [[36]](#bookmark42) | EA |  |  | Insr, CO |  |
| Zhao, Wang and Wang (2022) [[37]](#bookmark43) | ABC | Q-learning\*1 | Ls | Insr, Swap, B-Insr, B-Swap |  |
| Zhao, Jiang and Wang (2022) [[38]](#bookmark44) | MBO | Q-learning\*2 | Ls | Insr, Swap, 2p-shuf, 2p-Revr |  |
| Zhao, Zhou and Wang (2023) [[39]](#bookmark45) | SS | Q-learning\*1 | Ls | Insr, Swap, B-Insr, B-Swap |  |
| Shao, Shao and Pi (2023) [[40]](#bookmark46) | ILS |  |  | Insr, Swap |  |
| Yu, Gao, Ma and Pan (2023) [[41]](#bookmark47) | ABC, PSO, GA, Jaya | Q-learning\*1 | Ls | Insr, Swap, DC |  |
| Zhao, Zhou, Xu, Zhu and Jonrinaldi (2023) [[42]](#bookmark48) | SS | Q-learning\*1 | Opt | Insr, Swap, B-Insr |  |
| Zhao, Di and Wang (2023) [[43]](#bookmark49) | HH | Q-learning\*1 | Heu | Insr, Swap |  |
| Zhang, Qian, Hu and Yang (2023) [[44]](#bookmark50) | HH,EA | Q-learning\*1 | Heu | Insr, Swap, 2p-Revr |  |
| Jia, Yan and Wang (2023) [[45]](#bookmark51) | MA | Q-learning\*1 | Od | Insr, Swap, CO |  |
| Feng, Zhao, Jiang, Tao and Mei (2024) [[1]](#bookmark3) | IGA |  |  | Insr | Customized rules |
| Han, Sang, Pan, Zhang and Guo (2024) [[46]](#bookmark52) | IGA |  |  | Swap | SA |
| **This work** | IGA | Sarsa\*2 | Opt,Ls,Od | Insr, Swap | Customized rules |

Notations: Algorithm - VNS: Variable Neighborhood Search, SS: Scatter Search, EDA: Estimation of Distribution Algorithm, MA: Memetic Algorithm, IGA: Iterated Greedy Algorithm, BBO: Biogeography-based Optimization, ICG: Iterated Cocktail Greedy, CRO: Chemical Reaction Optimization, BSA: Backtracking Bearch Algorithm, HH: Hyper Heuristic, EA: Evolutionary Algorithm, VND: Variable Neighborhood Descent, ABC: Artificial Bee Colony, GA: Genetic Algorithm, WWO: Water Wave Optimization, KDH: Knowledge-driven Constructive Heuristic, NMA: Network MA, BSO: Brain Storm Optimization, MBO: Migrating birds Optimization, ILS: Iterated Local Search, PSO: Particle Swarm Optimization; Learning (Learning mechanism) - ’\*n’: n RL algorithm(s) is(are) used; Decision (Action of the RL) - Opt: operators, Ls: local searches, Heu: heuristics, Od: other decisions; Local search - Insr: Insertion, Revr: Reverse, CO: Crossover, 1p-CO: 1-point CO, B-Insr: Block-based insertion, B-Swap: Block-based swap, 2p-shuf: 2-point shuffle, 2p-Revr: 2-point Reverse, B-Shift: Block-based shift, DC: Destruction-Construction; Acceptance: Geedy: Greedy Acceptance Criteria, Restart: Restart mechanisms, SA: Simulated Snnealing based Acceptance Criteria, Customized rules: Customized Acceptance Criteria.

**2 The proposed approach**

***2.1 IGA***

*2.1.1 Representation and initialization procedure*

For DPFSP, we use a very common representation of the solutions proposed by Naderi and Ruiz [2]. It is a set of $F$ lists, each of which is a sequence of jobs processed within a factory. Taking an example of an instance with 5 jobs and 2 factories, one possible expression of the solutions is [[2,3,5],[1,4]].

The initialization procedure consists of two steps: 1) Generate the initial solution using $NEH2\_{e}n$ [14], as presented in Algorithm S1; 2) Perform *local search*, i.e., $LocalSearch\_{IG}$ in Section 2.1.3, to improve the solution just obtained.

|  |
| --- |
| **Algorithm S1:** Pseudo code of $NEH2\_{en}\left(.\right)$ |
| **Output:** $π$$Job\_{list}\leftarrow $ sort all jobs according to their processing time in descending order$π\leftarrow ∅$ **For** $f=1$ **to** $F$ **do** $π\_{f}\leftarrow \{Job\_{list}\left[0\right]\}$ $π\leftarrow π∪π\_{f}$ $Job\_{list}\leftarrow Job\_{list}\Job\_{list}\left[0\right]$ // Remove the first job in $Job\_{list}$**End****While** $Job\_{list}\ne ∅$ **do** **For** $f=1$ **to** $F$ **do**Test job $Job\_{list}\left[0\right]$ in all possible position of $π\_{f}$ $C\_{max}^{f}$ is the lowest $C\_{max}$ obtained $p^{f}$ is the position where the lowest $C\_{max}$ is obtained **End**$f\_{min}\leftarrow arg\left(min\_{f=1}^{F}\left(C\_{max}^{f}\right)\right)$ Insert job $Job\_{list}\left[0\right]$ in $π\_{f\_{min}}$ at position $p^{f\_{min}}$ resulting in the lowest $C\_{max}$Extract at random job $h$ from position $p^{f\_{min}}-1$ or $p^{f\_{min}}+1$ from $π\_{f\_{min}}$Test job $h$ in all possible positions of $π\_{f\_{min}}$Insert job $h$ in $π\_{f\_{min}}$ at the position resulting in the lowest $C\_{max}$$Job\_{list}\leftarrow Job\_{list}\Job\_{list}\left[0\right]$ **End****Return** $π$ |

*2.1.2 Perturbation*

The *perturbation* consists of two steps: *destruction* and *construction*. We use the *destruction* employed by [14]. It randomly removes $d\_{IG}/2$ jobs from the critical factory (factory with the largest makespan) and the remaining $d\_{IG}/2$ jobs are randomly removed from the remaining $F-1$ factories, i.e., excluding the critical factory. As for the *construction*, we use the basic *construction* proposed by [47], which consumes less computation resources than the *construction* proposed by [14]. The procedure of the *construction* is that each job which is removed during the *destruction* is inserted into all possible positions in all factories of the partial solution and the best one with the smallest makespan is then the new partial solution.

*2.1.3 Local search*

The *local search* used in IGA (i.e., $LocalSearch\_{IG}$) is $LS3$ proposed by [14], which considers factory assignment and job sequencing simultaneously. The detailed procedure is presented in Algorithm S2.

|  |
| --- |
| **Algorithm S2:** Pseudo code of $LS3\left(.\right)$ |
| **Input:** $π$ // $π=\{π\_{1},\cdots ,π\_{F}\}$**Output:** $π$$C\_{max}^{\*}\leftarrow C\_{max}\left(π\right)$ $f\_{max}$ is a critical factory of $π$$cnt\leftarrow 0$ **While** $cnt<\left|π\_{f\_{max}}\right|$ **do** Randomly extract, without repetition, a job $j$ from position $k$ of $π\_{f\_{max}}$ **For** $f=1$ **to** $F$ **do**Test job $j$ in all positions of $π\_{f}$ $C\_{max}^{f}$ is the lowest $C\_{max}$ obtained $p^{f}$ is the position where the lowest $C\_{max}$ is obtained **End**$f\_{min}\leftarrow arg\left(min\_{f=1}^{F}\left(C\_{max}^{f}\right)\right)$ **If** $C\_{max}^{f}<C\_{max}^{\*}$ **then**Insert job $j$ to position $p^{f}$ of factory $f\_{min}$ $C\_{max}^{\*}\leftarrow C\_{max}^{f}$ $f\_{max}$ is a critical factory of $π$ $cnt\leftarrow 0$ **Else** Return job $j$ to position $k$ of $f\_{max}$ $cnt\leftarrow cnt+1$ **End****End****Return** $π$ |

*2.1.4 Acceptance*

As for $Accept\_{IG}$, our design is based on the following ideas, and it is detailed in Algorithm S3. Before the start of each episode, both $E\_{end}$ and $E\_{cou}$ are set to 0, and $E\_{flag}$ is fixed to $True$.

* As long as no worse solutions are accepted, $P$ increases until it is 1, which is controlled by step-size $s^{\*}$ and $Idx$.
* After the first acceptance ($E\_{cou}=1$) of a worse solution, the episode is not terminated as long as there are local improvements (i.e., $E\_{end}$ is set to 0).
* After the second acceptance ($E\_{cou}=2$), the episode is not terminated only if global improvements exist.
* Global improvements allow $Accept\_{IG}$ to be partly reset to the original state of the parameters ($E\_{flag},E\_{end}\leftarrow True,0$), but if worse solutions have already been accepted, $E\_{cou}$ is set to the state of the first acceptance ($E\_{cou}\leftarrow min\{E\_{cou},1\}$).
* Neither local nor global improvements can reset the $P$.

|  |
| --- |
| **Algorithm S3:** Pseudo code of $Accept\_{IG}\left(.\right)$ |
| **Input:** $π^{'}$(New solutions), $π$(Local solutions), $π^{\*}$(Global solutions), $C$, $C^{\*}$, $E\_{end}$, $E\_{cou}$, $Idx$**Output:** $π,π^{\*},C,C^{\*},E\_{flag},E\_{end},E\_{cou},Idx$**If** $C\_{max}\left(π^{'}\right)<C$ **then**$π\leftarrow π^{'},C\leftarrow C\_{max}\left(π^{'}\right)$ **If** $E\_{cou}=1$ **then**$E\_{end}\leftarrow 0$ **Elseif** $E\_{cou}=2$ **then**$E\_{flag}\leftarrow False$ **End** **If** $C<C^{\*}$ **then**$π^{\*}\leftarrow π,C^{\*}\leftarrow C$ $E\_{flag}\leftarrow True,E\_{end}\leftarrow 0,E\_{cou}\leftarrow \{E\_{cou},1\}$ **End****Elseif** $E\_{end}=1$ **then**$E\_{flag}\leftarrow False$**Else** // Accept based on probability curve **If** $rand\leq cbrt\left(s^{\*}⋅Idx\right)$ **then** // $s^{\*}\leftarrow 0.25$ $π\leftarrow π^{'},C\leftarrow C\_{max}\left(π^{'}\right)$ $E\_{end}\leftarrow 1,E\_{cou}\leftarrow E\_{cou}+1$ **Else** // If not accept, increase probability $Idx\leftarrow Idx+1$ **End****End****Return** $π,π^{\*},C,C^{\*},E\_{flag},E\_{end},E\_{cou},Idx$ |

***2.2 Memory mechanism***

*2.2.1 mixLS*

The $mixLS$ is detailed in Algorithm S4. In $mixLS$, the $flag$, decided by $d\_{M}$ ($flag\leftarrow d\_{M}\%2$), is used to control $swap$ (based on [5]) or $insert$ (based on [14]) to be used. And it is worth noting that $swap$ starts by randomly extracting a job from all jobs, rather than just from critical factories. After the *local search*, only the factories involved undergo $LS1$ (based on [2]). Finally, $mixLS$ only extracts part of jobs ($int\left(n/len\left(d\_{list}\right)\right)$, where $len\left(d\_{list}\right)$ is the number of perturbation operators in $d\_{list}$) in one call, and in it, each job is extracted randomly.

|  |
| --- |
| **Algorithm S4:** Pseudo code of $mixLS\left(.\right)$ |
| **Input:** $π,flag,d\_{list}$**Output:** $π$$counter\leftarrow 0$ $partJobs\leftarrow int\left(n/len\left(d\_{list}\right)\right)$**While** $counter\leq partJobs$ **do**$improve\leftarrow False$// Optimization between factories **If** $flag=0$ **then** // Based on $swap$ [5] Randomly extract a job $j$ from all jobs $Swap$ $j$ with any other possible jobs $C\_{max}^{swap}$ is the lowest $C\_{max}$ obtained $j^{'}$ is the job where $C\_{max}^{swap}$ is obtained $f\_{list}$ is a set of unduplicated factories where $j$ and $j^{'}$ are located I**f** $C\_{max}^{swap}<C\_{max}\left(π\right)$ **then**$π\leftarrow swap\left(j,j^{'}\right),improve\leftarrow True$ **End**  **Elseif** $flag=1$ **then** // Based on $LS3$ [14] Randomly extract a job $j$ from critical factory Test $j$ in all possible positions of $π$ $C\_{max}^{insert}$ is the lowest $C\_{max}$ obtained $f\_{list}$ is a set of unduplicated factories where $j$ and $p^{f}$ are located **If** $C\_{max}^{insert}<C\_{max}\left(π\right)$ **then**$π\leftarrow insert\left(p^{f},j\right),improve\leftarrow True$ **End** **End**// Optimization within factories **For** $f\in f\_{list}$ **do** // Based on $LS1$ [2] Randomly extract a job $j$ from factory $f$ Test $j$ in all possible positions of $π$ in factory $f$ $C\_{max}^{f}$ is the lowest $C\_{max}$ obtained $p^{f}$ is the position where $C\_{max}^{f}$ is obtained **If** $C\_{max}^{f}<C\_{max}\left(π\right)$ **then**$π\leftarrow insert\left(p^{f},j\right),improve\leftarrow True$ **End** **End** **If** $improve$ **then**$counter\leftarrow 0$ **Else**$counter\leftarrow counter+1$ **End****End****Return** $π$ |

*2.2.2 Acceptance of memory mechanism*

As for $Accept\_{Memory}$, our design is based on the following ideas, and it is detailed in the Algorithm S5.

* Only if the best solution in $OM$ outperforms the global solution, local/global solutions are updated simultaneously.
* If there is no global improvement, the local solution should avoid accepting the best solution in original $OM$ (i.e., the $OM$ just before $MemoryUpdate$).

|  |
| --- |
| **Algorithm S5:** Pseudo code of $Accept\_{Memory}\left(.\right)$ |
| **Input:** $OM,π,π^{\*},C^{\*},d\_{M}$**Output:** $π,π^{\*},C^{\*}$**If** $C\_{max}\left(OM\left[0\right]\right)<C^{\*}$ **then**$π\leftarrow OM\left[0\right]$ $π^{\*}\leftarrow π,C^{\*}\leftarrow C\_{max}\left(π\right)$**Else** **If** $d\_{M}<2$ **then**$π\leftarrow OM\left[1\right]$ **Else**$π\leftarrow OM\left[0\right]$ **End****End****Return** $π,π^{\*},C^{\*}$ |

***2.3 Learning mechanism***

*2.3.1 Sarsa*

The Q-value of Sarsa can be updated by (S1):

|  |  |
| --- | --- |
| $Q\left(s\_{t},a\_{t}\right)\leftarrow Q\left(s\_{t},a\_{t}\right)+α\left(r\_{t+1}+γQ\left(s\_{t+1},a\_{t+1}\right)\right)$  | (S1) |

where $Q\left(s\_{t},a\_{t}\right)$ represents the Q-value corresponding to action $a\_{t}$ taken in state $s\_{t}$; $r\_{t+1}$ represents the immediate reward obtained by taking action $a\_{t}$ in state $s\_{t}$; $Q\left(s\_{t+1},a\_{t+1}\right)$ is the expected Q-value of the action $a\_{t+1}$ selected in state $s\_{t+1}$.

*2.3.2* $ϵ$*-greedy/decay*

The $ϵ$-greedy strategy [48] is a good choice to make a balance between exploration and exploitation, as shown in (S2).

|  |  |
| --- | --- |
| $π\left(s\right)=\left\{\begin{array}{c}1-ϵ(|A(s)|-1)/|A(s)|, \&for greedy action\\ϵ/|A(s)|, \&for other actions\end{array}\right.$  | (S2) |

where $a$ is an action, $s$ is a state, $A\left(s\right)$ is the actions for state $s$, $\left|A\left(s\right)\right|$ is the number of actions for state $s$, $π\left(s\right)$ is the probability of choosing action $a$ for state $s$ and $ϵ\in \left[0,1\right]$. When $ϵ= 0$, it becomes a greedy strategy and there is less exploration but more exploitation. When $ϵ= 1$, it becomes a uniform distribution and there is more exploration but less exploitation. If there is a parameter $β$ that can reduce the value of $ϵ$ throughout the search process, $ϵ$-greedy strategy becomes $ϵ$-decay, namely, $ϵ\leftarrow ϵ⋅β$. It is worth mentioning that we also reduce the value of $γ$ throughout the search process, i.e., $γ\leftarrow γ⋅β$, based on the original $ϵ$-decay. The purpose of this design is to guide the algorithm from exploration to exploitation.

***2.4 Complexity analysis of the developed MLIGA***

The worst-case complexity of the proposed MLIGA is analyzed according to Algorithm 1. In the following, we provide the complexities of the functions involved in the algorithm:

Initialization: In the experiments performed in this work, the initial solution is generated using *NEH2\_en* heuristic. As claimed in [14], its worst-case complexity is $O\left(n^{2}mF\right)$, in which $n$ is the number of jobs, $m$ is the number of machines, and $F$ is the number of factories.

$LocalSearch\_{IG}$: The complexity of the employed local search is $O\left(n^{2}m\right)$.

$mixLS(.)$: As shown in Algorithm S4, there are two optimization methods. In the first method, there are two possibilities: when $flag$ is 0, the complexity is $O(2nm)$. When $flag$ is 1, the complexity is $O(nm)$. Therefore, the worst-case complexity of the first method is the maximum of them, i.e., $O(2nm)$. In the second method, the factorie(s) (up to two) affected in the last stage undergo a local search internally. Hence, its time complexity is $O(2nm/F)$. At last, these two methods will iterate $partJobs (int(n/len(d\\_\{list\})))$ times, in which $len\left(d\_{list}\right)$ is the number of actions in the first layer RL mechanism. Based on the discussion above, the complexity of $mixLS(.)$ is $O\left(\left(2n^{2}m+2n^{2}m/F\right)/len\left(d\_{list}\right)\right)$.

$perturbation(.)$: The perturbation mechanism consists of two parts: destruction and construction, whose complexities are $O\left(d\_{IG}\right)$ and $O\left(d\_{IG}nm/F\right)$ [48], respectively. Note that $d\_{IG}$ is the action of the first layer RL mechanism.

$brainstorm(.)$: As shown in Algorithm 2, two solutions are selected by using the binary tournament strategy in one iteration, requiring $O(2)$ computations. Since there are at most $F$ key factories, the worst-case complexity of the first $swap$ operation is $O(F)$. Similarly, given that there are at most $F-1$ non-critical factories, the second $swap$ operation’s complexity is $O(F-1)$. Thus, the complexity of $brainstorm(.)$ is $O\left(nm\left(2+F+F-1\right)/2\right)$.

$MemoryUpdate(.)$: Tournament selection requires $O\left(len\_{newOM}\right)$ computations, in which $len\_{newOM}$ represents the number of solutions in original memory set $OM$. Due to $mixLS(.)$ is repeated $index\left(d\_{IG}\right)+1$ times on two solutions, it requires $O\left(4\left(index\left(d\_{IG}\right)+1\right)\left(n^{2}m+n^{2}m/F\right)/len\left(d\_{list}\right)\right)$, where $index\left(d\_{IG}\right)$ is the index of $d\_{IG}$ in $d\_{list}$. Therefore, $MemoryUpdate(.)$ requires $O\left(4\left(index\left(d\_{IG}\right)+1\right)\left(n^{2}m+n^{2}m/F\right)/len\left(d\_{list}\right)+len\_{newOM}\right)$ computations.

*learning mechanism*: The time complexities of both layers of RL are $O(4)$, since the only required operation is to find a maximum Q-value from a list of actions and the number of actions in them both are four. Compared to other parts, the on-line learning time added by learning mechanism is negligible in the total running time of MLIGA.

In summary, the time complexity of MLIGA is $O\left(Iter\left(index\left(d\_{IG}\right)+1\right)\left(F+1\right)n^{2}m/F\right)$, where $Iter$ is the iterations’ number of steps 4-16 in Algorithm 1.

**3 Experimental design**

***3.1 Experimental setting***

*3.1.1 Benchmark*

As for the standard benchmarks, we used the ***Large test instances*** and ***Calibration instances*** used by [14]. The details of these two instances are as follows:

* ***Large test instances***: There are a total of 720 instances, divided into 6 groups according to the number of factories $F=\{2,3,4,5,6,7\}$. Each group has 120 instances of Taillard ranging from 20 jobs and 5 machines to 500 jobs and 20 machines.
* ***Calibration instances***: There are 50 different instances with $n$, $m$ and $F$ values, which are randomly sampled from above 720 instances. And their processing times are randomly generated.

*3.1.2 RPD*

The relative percentage deviation (RPD) is calculated as (S3). And the ARPD is the mean value of RPD.

|  |  |
| --- | --- |
| $RPD=\frac{C\_{max}-C\_{max}^{\*}}{C\_{max}^{\*}}×100$  | (S1) |

where $C\_{max}$ refers to the $C\_{max}$ obtained by an algorithm on an instance, and $C\_{max}^{\*}$ refers to the minimum value of $C\_{max}$ obtained by all compared algorithms on the same instance.

*3.1.3 Termination criterion*

As for the termination criteria, we used the maximum number of fitness evaluations proposed by [15]. To determine the maximum number of fitness evaluations, we used the method proposed by [48]. All comparison algorithms were tested on each instance and terminated once there was no significant improvements (i.e., the improvements during the last 10% of the search process divided by the total improvements so far is less than 0.01). The maximum number of fitness evaluations among all comparison algorithms was considered as the termination criterion for all algorithms [48]. It is worth noting that, considering the different initialization procedures of the comparison algorithms and to maintain fairness, the number of fitness evaluations in this work referred to the statistics of the main part of the algorithms, i.e., the number of fitness evaluations in the initialization procedures was not considered.

***3.2 Settings of ablation experiments***

*3.2.1 Initialization*

$NEH2$ and $RandInit$ are detailed in Algorithms S6 and S7, respectively. $RandInit$ inserts jobs from a random sequence of all jobs into a random position in a random factory in sequence.

|  |
| --- |
| **Algorithm S6:** Pseudo code of $NEH2\left(.\right)$ |
| **Output:** $π$$Job\_{list}\leftarrow $ sort all jobs according to their processing time in descending order$π\leftarrow ∅$ **For** $f=1$ **to** $F$ **do** $π\_{f}\leftarrow \{Job\_{list}\left[0\right]\}$ $π\leftarrow π∪π\_{f}$ $Job\_{list}\leftarrow Job\_{list}\Job\_{list}\left[0\right]$**End****While** $Job\_{list}\ne ∅$ **do** **For** $f=1$ **to** $F$ **do**Test job $Job\_{list}\left[0\right]$ in all possible position of $π\_{f}$ $C\_{max}^{f}$ is the lowest $C\_{max}$ obtained $p^{f}$ is the position where the lowest $C\_{max}$ is obtained **End**$f\_{min}\leftarrow arg\left(min\_{f=1}^{F}\left(C\_{max}^{f}\right)\right)$ Insert job $Job\_{list}\left[0\right]$ in $π\_{f\_{min}}$ at position $p^{f\_{min}}$ resulting in the lowest $C\_{max}$$Job\_{list}\leftarrow Job\_{list}\Job\_{list}\left[0\right]$ **End****Return** $π$ |

|  |
| --- |
| **Algorithm S7:** Pseudo code of $RandInit\left(.\right)$ |
| **Output:** $π$$Job\_{list}$a random sequence of all jobs$π\leftarrow ∅$ **For** $f=1$ **to** $F$ **do**$π\_{f}\leftarrow ∅$ $π\leftarrow π∪π\_{f}$ $Job\_{list}\leftarrow Job\_{list}\Job\_{list}\left[0\right]$**End****While** $Job\_{list}\ne ∅$ **do**Insert $Job\_{list}\left[0\right]$ to a random available position in a random factory $Job\_{list}\leftarrow Job\_{list}\Job\_{list}\left[0\right]$**End****Return** $π$ |

*3.2.2 Probability curve*

Inspired by some existing curves, we designed three other probability curves, namely, $linear(x)$, $sigmoid(x)$ and $prob(x)$, which are calculated by (S4)-(S6). Fig. S1 shows the difference between these four probability curves.

|  |  |
| --- | --- |
| $linear\left(x\right)=\left\{\begin{array}{c}x, \&0\leq x<1\\1, \&x\geq 1\end{array}\right.$  | (S4) |
| $sigmoid\left(x\right)=\left\{\begin{array}{c}0, x=0\\1/(1+e^{-12(x-0.5)}), 0<x<1\\1, x\geq 1\end{array}\right.$  | (S5) |
| $prob\left(x\right)=\left\{\begin{array}{c}0, x=0\\e^{-8\left(x-1\right)^{2}}, 0<x<1\\1, x\geq 1\end{array}\right.$  | (S6) |



Figure S1: Four probability curves

**4 Numerical and statistical results**

In this section, all the results were analyzed using the Analysis of Variance (ANOVA) technique and the Tukey test with 95% confidence intervals. Statistic indicators including DF, Adj SS, Adj MS, F-Value, and P-Value were obtained using the statistical software *Minitab* [49].

***4.1 Calibration of MLIGA***

In the calibration experiment, the results of analysis of variance are showed in Table S2. According to Table S2, the p-value is bigger than 0.050, which shows that there are no significant differences among these $d\_{list}$-values. In addition, for better discussion, Fig. S2 shows the mean ARPD values of these $d\_{list}$-values with Tukey’s Honest Significant Difference (HSD) 95% confidence intervals. Although the differences are not significant, there are some difference of mean ARPD values for these $d\_{list}$-values, which can be used to determine a specific value of $d\_{list}$. And for convenience in plotting, we use $d\_{min}-d\_{max}$ instead of $\left[d\_{min},\cdots ,d\_{max}\right]$.

Table S2:Result of analysis of variance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Source | DF | Adj SS | Adj MS | F-Value | P-Value |
| $$d\_{list}$$ | 20 | 0.535 | 0.02677 | 0.18 | 1.000 |
| Error | 1029 | 150.193 | 014596 |  |  |



Figure S2: Means plots of all $d\_{list}$-values. All means have Tukey's Honest Significant Difference (HSD) 95% confidence intervals.

***4.2 Effectiveness of MLIGA***

In the comparison experiment of the four state-of-the-art comparison algorithms and the MLIGA, the results of analysis of variance are showed in Table S3 and Fig. S3. Fig. S3 shows that: 1) The overall performance of the MLIGA is better than DABC, DFFO, QFOA, and QIGA; 2) Given that QIGA utilizes a single-layer RL framework, the results can also indicate that the learning mechanism proposed in this paper is superior to the single-layer RL framework. 3) The mean ARPD value of MLIGA is a little lower than TSIGA and there is not a big difference. In addition, according to Table S3, the p-value is less than 0.050, which can only show that not all means are equal, i.e., there must be a significant difference between at least two algorithms. For further analysis, we plotted Fig. S4 to show the result of Tukey simultaneous tests for differences of means, and if an interval does not contain zero, the corresponding means are significantly different. Based on the rule, only the difference between MLIGA and TSIGA is not significant in Fig. S4. We analyzed this phenomenon in more depth in the main article and found that the MLIGA performed significantly better in large-scale instances than TSIGA.

Table S3:Result of analysis of variance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Source | DF | Adj SS | Adj MS | F-Value | P-Value |
| $$Algorithm$$ | 20 | 12722 | 2544.43 | 527.78 | 0.000 |
| Error | 4314 | 20798 | 4.82 |  |  |



Figure S3: Means plots for all algorithms. All means have Tukey’s Honest Significant Difference (HSD) 95% confidence intervals.



Figure S4: Results of Tukey simultaneous tests for diferences of means.

***4.3 Ablation experiments***

*4.3.1 Initialization*

The results of analysis of variance are displayed in Table S4, which is divided into two parts: 1) The statistics on the quality of the initial solutions of different initialization procedures; 2) The statistics on the quality of the best solutions obtained of different initialization procedures. Table S4 shows that there are significant differences between at least two initialization procedures on both parts.

Table S3:Result of analysis of variance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Source | DF | Adj SS | Adj MS | F-Value | P-Value |
| **For initial solutions** |
| Initialization procedures | 2 | 101734 | 50866.9 | 2670.60 | 0.000 |
| Error | 2157 | 41084 | 19.0 |  |  |
| **For best solutions obtained** |
| Initialization procedures | 2 | 26.75 | 13.3731 | 65.20 | 0.000 |
| Error | 2157 | 442.45 | 0.2051 |  |  |

For further investigation, we plotted Fig. S5 about the initial solutions and Fig. S6 about the best solutions obtained. We can find that: 1) Compared to *RandInit*, *NEH2\_en* and *NEH2* can provide better initial solutions, but the difference between them is not significant; 2) The results of *NEH2\_en* are best. However, the advantage of *NEH2\_en* is not significant compared to *NEH2*; 3) Although the initial solutions generated by *RandInit* are poor, the best solutions obtained are not too bad. In summary, although the quality of the initial solutions affects the quality of the best solutions obtained, MLIGA does not rely much on the initial solutions, and due to the presence of forgetting in the memory mechanism, the worse initial solutions do not deteriorate MLIGA excessively.



Figure S5: Means plots of all initialization procedures about initial solutions and results of Tukey simultaneous tests for diferences of means. All means have Tukey’s Honest Significant Difference (HSD) 95% confidence intervals.



Figure S6: Means plots of all initialization procedures about best solutions obtained and results of Tukey simultaneous tests for diferences of means. All means have Tukey’s Honest Significant Difference (HSD) 95% confidence intervals.

*4.3.2 Probability curve*

As showed in Table S5, the differences among these probability curves are not significant. As shown in Fig. S7, $Accept\_{IG}$ is not sensitive to the choice of the probability curves, namely, $Accept\_{IG}$ has better robustness. Finally, although the differences are not significant, the $cbrt(x)$ has the smallest mean ARPD value, which indicates that it is still the best choice among the four curves.

Table S5:Result of analysis of variance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Source | DF | Adj SS | Adj MS | F-Value | P-Value |
| Probability curves | 3 | 0.058 | 0.01820 | 0.09 | 0.967 |
| Error | 2876 | 633.755 | 0.22036 |  |  |



Figure S7: Means plots of all probability curves and results of Tukey simultaneous tests for diferences of means. All means have Tukey’s Honest Significant Difference (HSD) 95% confidence intervals.

*4.3.3 Memory mechanism*

As showed in Table S6, the difference between MLIGA and memoryless MLIGA is significant. According to Fig. S8, this conclusion is further confirmed and the MLIGA is much better than memoryless MLIGA, which is sufficient to show that the memory mechanism has an important impact on the performance of MLIGA and the design of memory mechanism is successful.

Table S5:Result of analysis of variance

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Source | DF | Adj SS | Adj MS | F-Value | P-Value |
| With/without memory | 1 | 245.7 | 245.666 | 870.32 | 0.000 |
| Error | 1438 | 405.9 | 0.282 |  |  |



Figure S8: Means plots of the MLIGA with or without memory and results of Tukey simultaneous tests for diferences of means. All means have Tukey’s Honest Significant Difference (HSD) 95% confidence intervals.

**References**

[1] X. Feng, F. Zhao, G. Jiang, T. Tao, and X. Mei, “A tabu memory based iterated greedy algorithm for the distributed heterogeneous permutation flowshop scheduling problem with the total tardiness criterion,” Expert Systems with Applications, p. 121790, 2023.

[2] B. Naderi and R. Ruiz, “The distributed permutation flowshop schedul- ing problem,” Computers & Operations Research, vol. 37, no. 4, pp. 754–768, 2010.

[3] B. Naderi and R. Ruiz, “A scatter search algorithm for the distributed permutation flowshop scheduling problem,” European Journal of Operational Research, vol. 239, no. 2, pp. 323–334, 2014.

[4] S.-Y. Wang and L. Wang, “An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flowshop scheduling problem,” IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 46, no. 1, pp. 139–149, 2015.

[5] V. Fernandez-Viagas and J. M. Framinan, “A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem,” International Journal of Production Research, vol. 53, no. 4, pp. 1111–1123, 2015.

[6] J. Lin and S. Zhang, “An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flowshop scheduling problem,” Computers & Industrial Engineering, vol. 97, pp. 128–136, 2016.

[7] S.-W. Lin and K.-C. Ying, “Minimizing makespan for solving the dis- tributed no-wait flowshop scheduling problem,” Computers & Industrial Engineering, vol. 99, pp. 202–209, 2016.

[8] H. Bargaoui, O. B. Driss, and K. Gh dira, “A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion,” Computers & Industrial Engineering, vol. 111, pp. 239–250, 2017.

[9] J. Deng and L. Wang, “A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem,” Swarm and evolutionary computation, vol. 32, pp. 121–131, 2017.

[10] J. Lin, Z.-J. Wang, and X. Li, “A backtracking search hyper-heuristic for the distributed assembly flowshop scheduling problem,” Swarm and evolutionary computation, vol. 36, pp. 124–135, 2017.

[11] W. Shao, D. Pi, and Z. Shao, “Optimization of makespan for the distributed no-wait flow shop scheduling problem with iterated greedy algorithms,” Knowledge-Based Systems, vol. 137, pp. 163–181, 2017.

[12] V. Fernandez-Viagas, P. Perez-Gonzalez, and J. M. Framinan, “The distributed permutation flow shop to minimise the total flowtime,” *Computers & Industrial Engineering*, vol. 118, pp. 464–477, 2018.

[13] Q.-K. Pan, L. Gao, L. Xin-Yu, and F. M. Jose, “Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem,” Applied Soft Computing, vol. 81, p. 105492, 2019.

[14] R. Ruiz, Q.-K. Pan, and B. Naderi, “Iterated greedy methods for the distributed permutation flowshop scheduling problem,” Omega, vol. 83, pp. 213–222, 2019.

[15] J.-j. Wang and L. Wang, “An iterated greedy algorithm for distributed hybrid flowshop scheduling problem with total tardiness minimization,” in 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE), 2019, pp. 350–355.

[16] T. Meng, Q.-K. Pan, and L. Wang, “A distributed permutation flowshop scheduling problem with the customer order constraint,” Knowledge-Based Systems, vol. 184, p. 104894, 2019.

[17] Z. Shao, W. Shao, and D. Pi, “Effective heuristics and metaheuristics for the distributed fuzzy blocking flowshop scheduling problem,” Swarm and Evolutionary Computation, vol. 59, p. 100747, 2020.

[18] Q.-K. Pan, L. Gao, and L. Wang, “An effective cooperative co-evolutionary algorithm for distributed flowshop group scheduling problems,” IEEE Transactions on Cybernetics, vol. 52, no. 7, pp. 5999–6012, 2022.

[19] J.-P. Huang, Q.-K. Pan, and L. Gao, “An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times,” Swarm and Evolutionary Computation, vol. 59, p. 100742, 2020.

[20] F. Zhao, R. Ma, and L. Wang, “A self-learning discrete jaya algorithm for multiobjective energy-efficient distributed no-idle flowshop scheduling problem in heterogeneous factory system,” IEEE Transactions on Cybernetics, vol. 52, no. 12, pp. 12 675–12 686, 2021.

[21] Y.-Z. Li, Q.-K. Pan, J.-Q. Li, L. Gao, and M. F. Tasgetiren, “An adaptive iterated greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems,” Swarm and Evolutionary Computation, vol. 63, p. 100874, 2021.

[22] Z. Shao, W. Shao, and D. Pi, “Effective constructive heuristic and iterated greedy algorithm for distributed mixed blocking permutation flow-shop scheduling problem,” Knowledge-Based Systems, vol. 221, p. 106959, 2021.

[23] J.-y. Mao, Q.-k. Pan, Z.-h. Miao, and L. Gao, “An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance,” Expert Systems with Applications, vol. 169, p. 114495, 2021.

[24] Z.-Q. Zhang, B. Qian, R. Hu, H.-P. Jin, and L. Wang, “A matrix-cube-based estimation of distribution algorithm for the distributed assembly permutation flow-shop scheduling problem,” Swarm and Evolutionary Computation, vol. 60, p. 100785, 2021.

[25] H. Li, X. Li, and L. Gao, “A discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem,” Applied Soft Computing, vol. 100, p. 106946, 2021.

[26] W. Shao, Z. Shao, and D. Pi, “Multi-objective evolutionary algorithm based on multiple neighborhoods local search for multi-objective distributed hybrid flow shop scheduling problem,” Expert systems with applications, vol. 183, p. 115453, 2021.

[27] Y.-Z. Li, Q.-K. Pan, K.-Z. Gao, M. F. Tasgetiren, B. Zhang, and J.-Q. Li, “A green scheduling algorithm for the distributed flowshop problem,” Applied Soft Computing, vol. 109, p. 107526, 2021.

[28] S. Chen, Q.-K. Pan, L. Gao, and H. yan Sang, “A population-based iterated greedy algorithm to minimize total flowtime for the distributed blocking flowshop scheduling problem,” Engineering Applications of Artificial Intelligence, vol. 104, p. 104375, 2021.

[29] Y.-Y. Huang, Q.-K. Pan, J.-P. Huang, P. N. Suganthan, and L. Gao, “An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem,” Computers & Industrial Engineering, vol. 152, p. 107021, 2021.

[30] F. Zhao, J. Zhao, L. Wang, and J. Tang, “An optimal block knowledge driven backtracking search algorithm for distributed assembly no-wait flowshop scheduling problem,” Applied Soft Computing, vol. 112, p. 107750, 2021.

[31] F. Zhao, L. Zhang, J. Cao, and J. Tang, “A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem,” Computers & Industrial Engineering, vol. 153, p. 107082, 2021.

[32] Y. Yang and X. Li, “A knowledge-driven constructive heuristic algorithm for the distributed assembly blocking flow shop scheduling problem,” Expert Systems with Applications, vol. 202, p. 117269, 2022.

[33] Y.-Z. Li, Q.-K. Pan, X. He, H.-Y. Sang, K.-Z. Gao, and X.-L. Jing, “The distributed flowshop scheduling problem with delivery dates and cumulative payoffs,” Computers & Industrial Engineering, vol. 165, p. 107961, 2022.

[34] W. Shao, Z. Shao, and D. Pi, “A network memetic algorithm for energy and labor-aware distributed heterogeneous hybrid flow shop scheduling problem,” Swarm and Evolutionary Computation, vol. 75, p. 101190, 2022.

[35] Y. Hou, Y. Fu, K. Gao, H. Zhang, and A. Sadollah, “Modelling and optimization of integrated distributed flow shop scheduling and distribution problems with time windows,” Expert Systems with Applications, vol. 187, p. 115827, 2022.

[36] G. Zhang, B. Liu, L. Wang, D. Yu, and K. Xing, “Distributed coevolutionary memetic algorithm for distributed hybrid differentiation flowshop scheduling problem,” IEEE Transactions on Evolutionary Computation, vol. 26, no. 5, pp. 1043–1057, 2022.

[37] F. Zhao, Z. Wang, and L. Wang, “A reinforcement learning driven artificial bee colony algorithm for distributed heterogeneous no-wait flowshop scheduling problem with sequence-dependent setup times,” IEEE Transactions on Automation Science and Engineering, 2022.

[38] F. Zhao, T. Jiang, and L. Wang, “A reinforcement learning driven cooperative meta-heuristic algorithm for energy-efficient distributed no-wait flowshop scheduling with sequence-dependent setup time,” IEEE Transactions on Industrial Informatics, 2022.

[39] F. Zhao, G. Zhou, and L. Wang, “A cooperative scatter search with reinforcement learning mechanism for the distributed permutation flowshop scheduling problem with sequence-dependent setup times,” IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2023.

[40] W. Shao, Z. Shao, and D. Pi, “Modelling and optimization of distributed heterogeneous hybrid flow shop lot-streaming scheduling problem,” Expert Systems with Applications, vol. 214, p. 119151, 2023.

[41] H. Yu, K.-Z. Gao, Z.-F. Ma, and Y.-X. Pan, “Improved meta-heuristics with Q-learning for solving distributed assembly permutation flowshop scheduling problems,” Swarm and Evolutionary Computation, vol. 80, p. 101335, 2023.

[42] F. Zhao, G. Zhou, T. Xu, N. Zhu et al., “A knowledge-driven cooperative scatter search algorithm with reinforcement learning for the distributed blocking flow shop scheduling problem,” Expert Systems with Applications, vol. 230, p. 120571, 2023.

[43] F. Zhao, S. Di, and L. Wang, “A hyperheuristic with Q-learning for the multiobjective energy-efficient distributed blocking flow shop scheduling problem,” IEEE Transactions on Cybernetics, 2022.

[44] Z.-Q. Zhang, B. Qian, R. Hu, and J.-B. Yang, “Q-learning-based hyper-heuristic evolutionary algorithm for the distributed assembly blocking flowshop scheduling problem,” Applied Soft Computing, vol. 146, p. 110695, 2023.

[45] Y. Jia, Q. Yan, and H. Wang, “Q-learning driven multi-population memetic algorithm for distributed three-stage assembly hybrid flow shop scheduling with flexible preventive maintenance,” Expert Systems with Applications, vol. 232, p. 120837, 2023.

[46] Q.-Y. Han, H.-Y. Sang, Q.-K. Pan, B. Zhang, and H.-W. Guo, “An efficient collaborative multi-swap iterated greedy algorithm for the distributed permutation flowshop scheduling problem with preventive maintenance,” Swarm and Evolutionary Computation, vol. 86, p. 101537, 2024.

[47] K.-C. Y. Shih-Wei Lin and C.-Y. Huang, “Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm,” International Journal of Production Research, vol. 51, no. 16, pp. 5029–5038, 2013.

[48] M. Karimi-Mamaghan, M. Mohammadi, B. Pasdeloup, and P. Meyer, “Learning to select operators in meta-heuristics: An integration of Q-learning into the iterated greedy algorithm for the permutation flowshop scheduling problem,” European Journal of Operational Research, vol. 304, no. 3, pp. 1296–1330, 2023.

[49] A. Alin, “Minitab,” Wiley interdisciplinary reviews: computational statistics, vol. 2, no. 6, pp. 723–727, 2010.