A Novel General Variable Neighborhood Search through Q-Learning for No-Idle Flowshop Scheduling

Hande Öztop
Department of Industrial Engineering
Yasar University
Izmir, Turkey
hande.oztop@yasar.edu.tr

Mehmet Fatih Tasgetiren
Department of International Logistics Management
Yasar University
Izmir, Turkey
fatih.tasgetiren@yasar.edu.tr

Levent Kandiller
Department of Industrial Engineering
Yasar University
Izmir, Turkey
levent.kandiller@yasar.edu.tr

Quan-Ke Pan
School of Mechatronic Engineering and Automation
Shanghai University
Shanghai, China
panquanke@shu.edu.cn

***Abstract*— In this study, a novel general variable neighborhood search through Q-learning (GVNS-QL) algorithm is proposed to solve the no-idle flowshop scheduling problem with the makespan objective. In the outer loop of the GVNS-QL, insertion, and exchange operators are used to shaking the permutation. On the other hand, in the inner loop of variable neighborhood descent procedure, variable iterated greedy and variable block insertion heuristic algorithms are employed with two effective insertion local search procedures. The proposed GVNS-QL defines the parameters of the algorithm using a Q-learning mechanism. The developed GVNS-QL algorithm is compared with the traditional iterated greedy (IG) algorithm using the well-known benchmark set. The comprehensive computational experiments show that the GVNS-QL outperforms the traditional IG algorithm. The results of the IG and GVNS-QL algorithms are also compared with the current best-known solutions reported in the literature. The computational results show that the proposed GVNS-QL algorithm improves the current best-known solutions for 104 out of 250 instances.**

Keywords— no-idle flowshop scheduling problem; makespan; general variable neighborhood search; Q-learning; variable iterated greedy; variable block insertion.

I. Introduction

In a flowshop, a set of *n* jobs are processed on *m* serial machines following the same route, generally, machine *1*, machine *2*, …, machine *m*. It is generally assumed that job pre-emption is not allowed and, all machines and jobs are ready at time zero. In the Permutation Flowshop Scheduling Problem (PFSP), once a job order (permutation) is determined on the first machine, this job order is employed for all machines, i.e., each machine processes the jobs with the same job permutation. Then, the PFSP aims to find the job permutation that optimizes a given performance criterion. The PFSP is well-known to be NP-hard [1].

In this study, we focus on an extension of the PFSP, in which idle time is not permitted between the jobs on the machines. This variant of the PFSP is known as the No-Idle Flowshop Scheduling Problem (NIFSP). In many real production environments such as foundries, integrated circuits, and fiberglass, once the machines start to process the jobs, the idle time is undesirable, as expensive machines are used. In this paper, we study the *m*-machine (Fm) no-idle permutation flowshop scheduling problem with the makespan (C_{max}) objective, namely, $Fm|prmu, no-idle|C_{max}$. Accordingly, the goal is to obtain the best job permutation that minimizes the makespan (maximum completion time). The NIFSP has also been proven to be NP-hard [2].

Many exact and heuristic solution approaches have been proposed to solve the NIFSP. Vachajitpan [3] developed a mixed-integer programming model and a Branch & Bound (B&B) algorithm for the NIFSP with the makespan criterion. Afterward, a B&B approach was also developed for the NIFSP by Saadani et al. [4]. Since these B&B methods can only be used to solve small-sized problems, heuristic methods have been generally addressed to solve the NIFSP. Adiri and Pohoryles [5] developed a polynomial-time heuristic method to solve the NIFSP with two machines considering the total completion time criterion, and revealed that *2*-machine no-idle PFSP and *2*-machine PFSP are the same for the makespan objective. The NIFSP with the makespan criterion was formulated as an asymmetric traveling salesman problem in [6], where the authors presented the nearest insertion rule-based heuristic method. An efficient constructive heuristic was also presented by Kalczynski and Kamburowski [7] for the NIFSP with the makespan objective. Later, a two-stage improved greedy algorithm was presented for the same problem [8].

Furthermore, discrete differential evolution (DDE) and hybrid discrete particle swarm optimization (HDPSO) algorithms were proposed for the NIFSP with the makespan objective [9, 10]. In these two papers, a speed-up approach was developed for the insertion neighborhood to decrease the time complexity from $O(n^3m)$ to $O(n^2m)$. Ruiz et al. [11] assessed the performance of the iterated greedy (IG) algorithm to solve the NIFSP with the makespan objective, and according to their results, the IG outperforms the DDE and the HDPSO. The benchmark instances were also presented for the NIFSP in [11]. A hybrid discrete differential evolution (HDDE) algorithm was also developed for the same problem by [12], and according to their results, the HDDE outperforms the DDE, HDPSO, and IG algorithms. A variable iterated greedy algorithm with differential evolution was also developed by Tasgetiren et al. [13] for the NIFSP with the makespan and total flowtime criteria. Later, an invasive weed optimization (IWO) algorithm was presented by [14] for the NIFSP, and their results, which are based on the benchmark set of [15], show that the IWO outperforms the IG and HDPSO. Recently, a memetic algorithm was developed by [16] for the NIFSP with the makespan objective, where the authors compared their algorithm with the other well-known heuristics from the literature. They improved 89 out of the 250 best solutions presented for the benchmark instances of [11].

Additionally, Tasgetiren et al. [17] presented a discrete artificial bee colony algorithm and Shao et al. [18] developed a hybrid discrete teaching-learning based metaheuristic for the NIFSP with the total tardiness criterion. As other extensions of the NIFSP, a two-stage memetic algorithm was proposed for the distributed NIFSP by [19]. Later, a mathematical model and heuristic algorithms were developed for the mixed NIFSP with sequence-dependent setup times in [20].

Mladenovic and Hansen [21] presented a Variable Neighborhood Search (VNS) by employing changes in the neighborhood systematically. Afterward, the VNS was extended by Hansen et al. [22] as a General Variable Neighborhood Search (GVNS) algorithm. The GVNS has been effectively applied to solve a variety of problems such as single machine scheduling problem [23], the NIFSP [24], distributed PFSP [25] and distributed no-wait flowshop scheduling problem [26]. Note that, insertion, swap, IG and iterated local search algorithms are used in the GVNS of [24].

Inspired by the abovementioned effective applications of the GVNS in various scheduling problems, this study presents a novel GVNS algorithm, called as GVNS-QL, for the NIFSP with the makespan criterion by incorporating a Q-learning mechanism. In the developed GVNS-QL, the initial solution is obtained by using the FRB5 constructive heuristic. Insertion and exchange operators are used in the outer loop, whereas an effective Variable IG (VIG_{ALL}) and an effective Variable Block Insertion Heuristic (VBIH) are used in the inner loop of the Variable Neighborhood Descent (VND) phase.

The Q-learning (QL) is one of the well-known reinforcement learning algorithms. The QL aims to choose an appropriate action based on experience. In the QL, once the learner performs a chosen action, it obtains a reward or penalty. Then, it learns to choose the best action to perform by assessing the action alternatives using the cumulative rewards (Q-values). We use the QL approach to choose the parameters of the algorithm. Namely, parameters are determined through a Q-learning approach in the proposed GVNS-QL, instead of using constant parameter values. We compare the performance of the developed GVNS-QL algorithm with the well-known IG algorithm using the benchmarks from the literature. Then, we compare the results of these algorithms with the current best-known solutions reported in the literature. The rest of the paper is organized as follows. In Section 2, the NIFSP is explained formally. In Section 3, the developed GVNS-QL algorithm is described. In Section 4, computational results are presented. Finally, in Section 5, conclusions and future research directions are provided.

II. Problem Definition

The NIFSP can be described as follows: a set of n jobs $J = \{1,2,\dots,n\}$ must be processed on a set of m machines $M = \{1,2,\dots,m\}$ in the same order. Each job has m operations, where k^{th} operation of job j must be processed on machine k with a given processing time p_{jk} without an interruption. Each job can be processed by only one machine and each machine can process only one job, at a time. All machines process the jobs with the same job permutation. Idle time is not allowed between two subsequent job operations on the same machine. All jobs are ready at the beginning. The goal is to obtain the best job permutation that minimizes the makespan.

Let $\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$ represent the job permutation and $\pi_j^E = \{\pi_1, \pi_2, .., \pi_j\}$ represent a partial order of π such that $1 < j < n$. Additionally, $F\left(\pi_j^E, k, k+1\right)$ denotes the minimum difference between the completion time of processing the last job of π_j^E on machines $k+1$ and k, which is limited by the no-idle constraint. Consequently, the makespan C_{max} can be calculated as follows, where $p_{\pi_j,k}$ denotes the processing time of job π_j on machine k:

$$F(\pi_1^E, k, k+1) = p_{\pi_1,k+1} \quad k = 1,2,\dots,m-1 \tag{1}$$

$$F\left(\pi_j^E, k, k+1\right) = \max\{F\left(\pi_{j-1}^E, k, k+1\right) - p_{\pi_j,k}, 0\} + p_{\pi_j,k+1} \quad j = 2,3,..,n \quad k = 1,2,\dots,m-1 \tag{2}$$

$$C_{max} = \sum_{k=1}^{m-1} F(\pi_n^E, k, k+1) + \sum_{j=1}^{n} p_{\pi_j,1} \tag{3}$$

III. General Variable Neighborhood Search Algorithm Through Q-Learning

The VNS is an effective heuristic procedure that uses a multi neighborhood structure during the search. The VNS has two core phases: (1) shaking phase that perturbs the solution to escape from local optima, and (2) local search phase that explores the neighborhood of the solution by employing the given neighborhood structures. The VNS has a set N_k of neighborhood structures, where $k = 1,2,..,k_{max}$. In the VNS, the solution is initialized randomly or using a constructive heuristic. Then, shaking and local search phases are employed on the solution until the stopping criterion is satisfied, where the stopping criterion can be defined as the maximum CPU time or the maximum number of iterations. Later, an extended version of the VNS, named as GVNS, was proposed by [22]. In the GVNS, the local search phase of the VNS is replaced with a VND algorithm, which is a deterministic version of the VNS, where the change of neighborhoods is performed in a deterministic way.

In this study, we propose a novel GVNS algorithm through Q-learning, i.e. GVNS-QL, where the algorithm parameters are determined through a Q-learning approach. Namely, values of the following parameters are updated through the GVNS procedure using a QL mechanism: ϵ (ϵjumping probability), τP (parameter of the acceptance criterion), d_{max} (maximum destruction/block size), lf (learning factor of the Q-learning function) and df (discount factor of the Q-learning function). In the QL mechanism, once an action (parameter setting) is performed, a reward or penalty is obtained for that action. Then, the algorithm learns to choose the best action to perform for each parameter by assessing the action alternatives based on their cumulative rewards (Q-values). The details of the QL strategy are explained in Section III.C.

The main framework of the proposed GVNS-QL algorithm is outlined in Fig. 1, where $U\,(0,1)$ is a uniform random number in between 0 and 1. As shown in Fig.1, the GVNS-QL has two main neighborhood parameters: k_{max} the number of neighborhoods employed in the outer loop and q_{max} the number of neighborhoods used in the inner (VND) loop. In this study, we set $q_{max} = 2$ and $k_{max} = 2$. As shown in Fig. 1, the initial solution is obtained by the FRB5 constructive heuristic [27], which is an extended version of the NEH heuristic [28]. Then, the initial parameter values of the algorithm are determined randomly. In the shaking part, insertion and exchange operators are employed in the outer loop. For the inner loop of VND, two powerful algorithms, namely, VIG_{ALL} and VBIH algorithms are used. In the proposed GVNS-QL, if the new solution is better than the incumbent solution, it is accepted and the Q-values of the performed actions are updated in the Q-value table for the parameters according to a Q-learning function. Otherwise, a simulated annealing-type acceptance criterion [29] with a temperature T is employed to decide whether the new permutation is accepted or not. T is calculated by equation (4), where τP is a parameter to be adjusted:

$$T = \frac{\sum_{j=1}^{n}\sum_{k=1}^{m} p_{jk}}{10nm} \times \tau P \tag{4}$$

$GVNS-QL$

$\pi = FRB5, \pi^{best} = \pi, k_{max} = 2$
$Do\{$
  $Initialize\ actions\ of\ parameters\ randomly\ from\ action\ list$
  $k = 1$
  $Do\{$
    $If\ (k = 1)\ then\ \pi^1 = Insertion(\pi)$
    $If\ (k = 2)\ then\ \pi^1 = Exchange(\pi)$
    $\pi^2 = VND(\pi^1)$
    $If\ f(\pi^2) < f(\pi)$
      $\pi = \pi^2$
      $k = 1$
      $Update\ the\ Q\text{-}value\ table\ with\ a\ reward\ of\ (1/f(\pi))$
    $Else$
      $k = k + 1$
      $If\ \left(U(0,1) < exp\{-(f(\pi^2) - f(\pi))/T\}\right)$
        $\pi = \pi^2$
      $Endif$
    $Endif$
  $\}While(k \le k_{max})$
  $If\ \left(f(\pi) < f(\pi^{best})\right)$
    $\pi^{best} = \pi$
  $Endif$
$\}While(NotTermination)$
$Return\ \pi^{best}$

Fig. 1. GVNS through Q-learning

The VND algorithm of the proposed GVNS-QL is explained in Fig. 2. As shown in Fig. 2, the parameter values are selected at each iteration using a QL strategy. Namely, the actions are determined for the parameters either randomly with a jumping probability ϵ or according to the Q-values of the actions, i.e., the actions with the maximum Q-values are selected. As seen in Fig. 2, VIG_{ALL} and VBIH algorithms are employed in the VND. Then, a similar acceptance procedure as in the main GVNS-QL is employed, where Q-values of the performed actions are also updated for the parameters according to a Q-learning function. FRB5 constructive heuristic, VIG_{ALL}, VBIH and QL procedures are explained in the following subsections.

$VND\ (\pi)$

$q = 1, q_{max}$=2
$Do\{$
$If(U(0.1) < \epsilon)$
  $Determine\ actions\ of\ parameters\ randomly\ from\ action\ list$
$Else$
  $Determine\ actions\ of\ parameters\ according\ to\ Q\text{-}values$
  $in\ Q\text{-}value\ table$
$Endif$
 $If\ (q = 1)\ then\ \pi^1 = VIG_{ALL}(\pi, d_{max})$
 $If\ (q = 2)\ then\ \pi^1 = VBIH(\pi, d_{max})$
 $If\ f(\pi^1) < f(\pi)$
   $\pi = \pi^1$
   $q = 1$
   $Update\ the\ Q\text{-}value\ table\ with\ a\ reward\ of\ (1/f(\pi))$
 $Else$
   $q = q + 1$
   $If\ \left(U(0,1) < exp\{-(f(\pi^1) - f(\pi))/T\}\right)$
     $\pi = \pi^1$
   $Endif$
 $Endif$
$\}While(q \le q_{max})$
$Return\ \pi\ and\ f(\pi)$

Fig. 2. VND Algorithm

A. Constructive Heuristic

The developed GVNS-QL uses the well-known FRB5 heuristic [27] as a constructive heuristic. The FRB5 heuristic contains an extra local search compared to the NEH heuristic. In the FRB5 heuristic, initially, jobs are sorted in decreasing order of their total processing times and a partial solution is initialized with γ_1 similar to the NEH. Then, the rest of the jobs in γ are sequentially inserted into the partial solution, where an insertion local search is employed on the partial solution at each iteration. The FRB5 heuristic is explained in Fig. 3.

$FRB5$ Heuristic

$\gamma = DecreasingOrder\left(\sum_{k=1}^{m} p_{jk}\right)$
$\pi_1 = \gamma_1$
$for\ z = 2\ to\ n\ do$
  $\pi = InsertJobInBestPosition(\pi, \gamma_z)$
  $\pi = LocalSearch(\pi)$
$endfor$
$return\ \pi\ and\ f(\pi)$

Fig. 3. FRB5 heuristic

B. VIG_{ALL} and VBIH Algorithms

Recently, the traditional IG [30] has been successfully extended to an IG_{ALL} algorithm by [31]. Similar to the IG algorithm, a destruction-construction procedure is employed in the IG_{ALL} as follows. In the destruction part, d jobs are randomly removed from the permutation π and kept in π^d, while the rest of the jobs are kept in π^p. In the construction part, the removed jobs in π^d are sequentially inserted into the partial solution π^p according to the best insertion approach. Unlike the IG, the IG_{ALL} applies an insertion local search (Fig. 4) to the partial solution before the construction part, as long as the solution is improved. As shown in Fig. 4, the insertion local search removes the job π_j from the solution π randomly, and inserts it into all positions of π. Once the best insertion is found, the job π_j is inserted into that position. These steps are repeated for all jobs. If an improvement is found, the local search is restarted until no improving solution is generated.

```
Insertion Local Search (π)
for j = 1 to n do
    π* = InsertJobInBestPosition(π, π_j)
    if f(π*) < f(π)
        π = π*
    endif
endfor
return π and f(π)
```

Fig. 4. Insertion local search

In this paper, we propose a variable IG_{ALL} algorithm, named as VIG_{ALL}, as one of the strategies of the VND algorithm. As shown in Fig. 5, the VIG_{ALL} algorithm takes the solution and the maximum destruction size (d_{max}) from the VND procedure, where the d_{max} parameter is determined through a QL mechanism. In the beginning, the destruction size is set as $d = 1$, and increased by one if the solution is not improved until the d reaches at d_{max}. Note that, if the solution improves at any d value, the destruction size d is reset to one. The VIG_{ALL} algorithm applies destruction-construction and local search procedures to the solution, respectively. While the insertion local search is employed on the partial solution π^p as long as it is improved, the referenced insertion scheme (RIS) and insertion local search procedures are employed on the complete solution π^1 in another VND loop, i.e. $Local\ Search\_VND$ (Fig. 7).

```
VIG_ALL(π, d_max)
d = 1
  do
      π^d, π^p = Destruction(π, d)
      π^p = Insertion Local Search (π^p)
      π^1 = Construction(π^d, π^p)
      π^2 = Local Search_VND(π^1)
      if (f(π^2) < f(π))
          π = π^2, d = 1
      else
          d = d + 1
      endif
  while(d ≤ d_max)
return π and f(π)
```

Fig. 5. VIG_{ALL} algorithm

Another strategy of the VND is the VBIH algorithm, which takes the solution and the maximum block size (d_{max}) from the VND procedure, where the d_{max} parameter is determined through a QL mechanism. The VBIH algorithm employs block insertion and local search procedures on a solution as shown in Fig. 6, where d denotes the block size. At the beginning, the block size is set as $d = 1$. In the VBIH algorithm, d consecutive jobs (π^d), i.e. block, are removed from the order π, where the rest of the jobs construct a partial solution (π^p). Then, an insertion local search (Fig. 4) is employed on the partial solution, as long as the solution is improved. Then, block insertion moves are applied to the partial solution and the best move is chosen. Afterward, similar to the VIG_{ALL} algorithm, two local search procedures are applied to the complete solution in a VND loop, i.e. $Local\ Search\_VND$, which is explained in Fig. 7. If the new solution after the local search procedures is better than the current one, the VBIH replaces the current solution and the block size is reset to one. Otherwise, the block size is incremented by one. These steps are reiterated until the maximum block size is reached.

```
VBIH(π, d_max)
d = 1
  do
      π^d, π^p = Remove block with d jobs from π
      π^p = Insertion Local Search (π^p)
      π^1 = InsertBlockInBestPosition(π^d, π^p)
      π^2 = Local Search_VND(π^1)
      if (f(π^2) < f(π)) then do
          π = π^2, d = 1
      else
          d = d + 1
      endif
  while(d ≤ d_max)
return π and f(π)
```

Fig. 6. VBIH algorithm

Local Search_VND procedure of the VIG_{ALL} and VBIH algorithms are outlined in Fig. 7. As seen in Fig. 7, the RIS and insertion local search procedures are employed in the Local Search_VND, where the RIS refers to a Referenced Insertion Scheme [32], which is explained in Fig. 8.

```
Local Search_VND (π)
u_max = 2 , u = 1
do{
   If (u = 1) then π^1 = RIS(π, π^best)
   If (u = 2) then π^1 = Insertion Local Search(π)
   If f(π^1) < f(π)
       π = π^1, u = 1
   else
       u = u + 1
   endif
}while(u ≤ u_max)
return π and f(π)
```

Fig. 7. Local Search_VND procedure

In the RIS, π^R represents the reference order, which is the best permutation obtained so far. As shown in Fig.8, the RIS chooses the first job in π^R and finds the best position for this job in the current solution π, by inserting it into all possible positions of the solution π. Then, it chooses the second job in π^R and finds the best position for this job in the current solution π. The iteration counter (c) is reset to one if an improvement occurs. Otherwise, it is incremented by one. These steps are applied until the iteration counter exceeds n.

```
RIS(π, π^best)
π^R = π^best, h = 1, c = 1
while(c ≤ n)do
    k = 1
    while (π_k ≠ π^R_h)
        k = k + 1
    end while
    h = (h + 1)(mod)n
    remove job π_k from π
    π^1 = insert job π_k into the best position in π
    if(f(π^1) < f(π))then
        π = π^1, c = 1
    else
        c = c + 1
    endif
endwhile
return π and f(π)
```

Fig. 8. RIS local search

C. Q-Learning Procedure

The Q-learning (QL) is one of the widely used reinforcement learning algorithms. The QL aims to choose an appropriate action based on experience by interacting with the environment. Once the agent (learner) performs a chosen action, it obtains a reward or penalty. Then, it learns to choose the best action to perform by assessing the action alternatives using the cumulative rewards (Q-values).

The Q-value can be calculated for each state-action pair by a Q-learning function given in equation (5) [33]. Then, Q-values are kept for all state-action pairs in a Q-value table. Let $S = [s_1, s_2, \dots s_p]$ be the set of states, $A = [a_1, a_2, \dots a_p]$ be the set of actions, r_{t+1} be the reward, $lf \in [0,1]$ be the learning factor, $df \in [0,1]$ be the discount factor and $Q(s_t, a_t)$ be the Q-value at time *t*. The learner aims to maximize its total reward.

$$Q_{t+1}(s_t, a_t) = Q(s_t, a_t) + lf[r_{t+1} + df * max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (5)$$

In the GVNS-QL algorithm, we assume that there is only one state for each parameter, where the reward is the $1/C_{max}$ value. As mentioned at the beginning of Section 3, we determine the ϵ, τP, d_{max}, lf and df parameters of the GVNS-QL algorithm using a QL approach. Namely, at each iteration, we update the Q-values of the chosen actions for the parameters through the Q-learning function. Then, in the next iteration, the algorithm chooses the best action (value) for each parameter with the maximum Q-value. Note that, in the GVNS-QL algorithm, we also choose the actions of the parameters randomly with a small ϵ jumping probability. The action list of each parameter is given in Table 1.

TABLE I. ACTION LIST OF THE PARAMETERS

Parameter	Action List
ϵ	{0.05, 0.10, 0.15, 0.20}
τP	{0.1, 0.2, 0.3, 0.4, 0.5}
d_{max}	{2, 3, 4, 5}
lf	{0.2, 0.4, 0.6, 0.8, 1}
df	{0.2, 0.4, 0.6, 0.8, 1}

D. IG Algorithm

The IG algorithm is one of the state-of-the-art heuristic algorithms in the literature, and it has been applied effectively to various scheduling problems such as permutation flowshops [30], hybrid flowshops [32, 34] and no-idle flowshops [11]. In this paper, we compare the proposed GVNS-QL algorithm with the well-known IG algorithm. In the traditional IG algorithm [30], the initial solution is generated using the NEH heuristic [28]. Afterward, a destruction-construction procedure is employed on the solution as follows. In the destruction part, d jobs are randomly removed from the permutation π and kept in π^d, and in the construction part, the removed jobs in π^d are sequentially inserted into the partial solution π^p. Then, an insertion local search is applied to the complete solution, as explained in Fig. 4. If an improvement is found, the local search is restarted until no improving solution is generated. Finally, the new solution is accepted using a simulated annealing-type acceptance criterion with a constant temperature T [29].

IV. COMPUTATIONAL RESULTS

In this paper, we present a novel GVNS-QL algorithm for the NIFSP with the makespan criterion. In order to assess the performance of the developed GVNS-QL, we use the benchmark set provided by [11] at http://soa.iti.es/rruiz. There are 250 problems in the set, containing all combinations of $n = \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $m = \{10, 20, 30, 40, 50\}$, where there are five problems for each combination of *n* and *m*.

In this paper, we compare the developed GVNS-QL algorithm with the well-known IG algorithm. Both IG and GVNS-QL algorithms are coded in C++ programming language on Microsoft Visual Studio 2013 and run on an Intel Core-i9 3.10 GHz computer with 32 GB memory. For each algorithm, five independent replications are conducted for each instance. In each replication, both algorithms are run for 30nm milliseconds, where n represents the number of jobs and m indicates the number of machines. In the IG algorithm, we use $d = 4$ and $\tau P = 0.4$ settings as proposed by the authors in [30]. For each algorithm, the relative percentage deviation (RPD) is computed for each instance by equation (6):

$$\text{RPD} = \frac{H-BS}{BS} \times 100 \quad (6)$$

where H indicates the makespan obtained by any of the heuristic algorithms for a given instance, and BS is the best-known result provided by [16]. The average RPD values over five replications are computed for each heuristic, along with the minimum, maximum, and standard deviation values. Afterward, the average relative percentage deviations (ARPD) are calculated for each instance group with the same instance size. Table 2 provides the ARPD values for each heuristic, classified by *n* and *m*, over five instances.

As shown in Table 2, the GVNS-QL outperforms the IG algorithm in terms of the maximum, minimum, and average ARPD values. The overall average ARPD value is 0.25% for the IG algorithm, whereas it is 0.09% for the GVNS-QL over 250 benchmark problems. In Table 2, the best average ARPD values among the two algorithms is denoted in bold for each instance set. The GVNS-QL has better average ARPD values than the IG algorithm for 42 out of 50 instance classes, where both algorithms have the same average ARPD values for the rest of the instance classes.

To evaluate the performance of the two algorithms statistically, we carried out a Wilcoxon signed-rank test with the significance level of α= 0.05, based on the results in Table 2. Let m_D denotes the median of the difference between the average ARPDs of the two algorithms. The null hypothesis H_0: $m_D = 0$ indicates that there is no statistically significant difference between the average ARPDs of the two compared algorithms and the alternative hypothesis $H_1 : m_D \neq 0$ indicates that there is a statistically significant difference between them. Since the *p*-value of the Wilcoxon signed-rank test is zero, it can be said that the difference between the IG and GVNS-QL algorithms is statistically significant at the α= 0.05 level. We also provide the interval plot of the two algorithms in Fig. 9. As shown in Fig. 9, the GVNS-QL is statistically better than the IG algorithm, since their confidence intervals do not overlap.

Furthermore, Table 3 reports the best results found by the heuristic algorithms for each instance, where the BS column represents the current best-known results provided by [16]. As seen in Table 3, the IG algorithm obtains a makespan result that is less than or equal to the current BS value for 129 instances. However, the GVNS-QL finds a makespan result that is less than or equal to the current BS value for 207 instances. The GVNS-QL improves the current BS results for 104 instances, whereas the IG improves the current BS results for 39 instances. In Table 3, the new best results found by the IG and GVNS-QL are emphasized in bold. Consequently, it can be said that the developed GVNS-QL is very effective to solve the NIFSP with the makespan criterion.

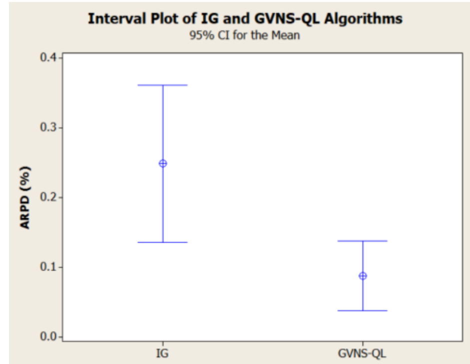

Fig. 9. Interval plot of the algorithms

TABLE II. COMPUTATIONAL RESULTS OF IG AND GVNS-QL

n	***m***	**IG**				**GVNS-QL**			
		ARPD (%)				**ARPD (%)**			
		Avg.	**Min.**	**Max.**	**Std. Dev.**	**Avg.**	**Min.**	**Max.**	**Std. Dev.**
50	10	0.01	-0.01	0.05	0.02	**-0.01**	-0.01	-0.01	0.00
	20	0.16	0.02	0.28	0.11	**0.06**	0.01	0.10	0.04
	30	0.40	0.06	0.82	0.31	**0.24**	0.14	0.34	0.09
	40	0.95	0.53	1.28	0.31	**0.41**	0.03	0.79	0.28
	50	2.59	1.75	3.28	0.61	**1.18**	0.92	1.57	0.27
100	10	0.07	0.01	0.13	0.06	**0.03**	0.02	0.08	0.03
	20	0.17	0.01	0.36	0.17	**0.03**	0.00	0.04	0.02
	30	0.15	-0.01	0.38	0.15	**0.05**	-0.03	0.22	0.11
	40	0.94	0.27	1.65	0.51	**0.26**	-0.01	0.64	0.28
	50	0.68	0.26	1.23	0.41	**0.17**	-0.03	0.46	0.20
150	10	**0.01**	0.01	0.01	0.00	**0.01**	0.01	0.01	0.00
	20	0.11	0.03	0.27	0.09	**0.02**	-0.01	0.07	0.03
	30	0.20	0.09	0.39	0.13	**0.04**	-0.01	0.15	0.07
	40	0.33	0.18	0.57	0.16	**0.11**	-0.04	0.26	0.13
	50	0.35	0.11	0.76	0.26	**0.09**	-0.01	0.27	0.12
200	10	**0.00**	0.00	0.00	0.00	**0.00**	0.00	0.00	0.00
	20	0.06	0.03	0.13	0.04	**0.03**	0.01	0.06	0.02
	30	0.29	0.01	0.52	0.22	**0.16**	0.08	0.31	0.10
	40	0.37	0.20	0.64	0.19	**0.12**	0.04	0.24	0.09
	50	0.40	0.12	0.72	0.24	**0.08**	-0.02	0.21	0.09
250	10	**0.00**	0.00	0.00	0.00	**0.00**	0.00	0.01	0.00
	20	0.06	0.01	0.14	0.06	**0.03**	0.00	0.08	0.04
	30	0.17	0.08	0.33	0.11	**0.08**	0.01	0.15	0.06
	40	0.21	0.02	0.42	0.15	**0.08**	0.01	0.16	0.06
	50	0.28	0.03	0.53	0.21	**0.09**	-0.01	0.22	0.10
300	10	**0.00**	0.00	0.00	0.00	**0.00**	0.00	0.00	0.00
	20	0.04	0.01	0.07	0.03	**0.01**	-0.01	0.02	0.01
	30	0.10	0.04	0.19	0.07	**0.03**	-0.01	0.06	0.03
	40	0.28	0.12	0.51	0.17	**0.09**	0.02	0.18	0.07
	50	0.43	0.11	0.82	0.28	**0.11**	0.01	0.24	0.10
350	10	**0.01**	0.01	0.03	0.01	**0.01**	0.00	0.01	0.00
	20	0.06	0.04	0.11	0.03	**0.01**	0.00	0.03	0.01
	30	0.08	0.01	0.14	0.05	**0.03**	-0.01	0.10	0.05
	40	0.27	0.14	0.41	0.12	**0.06**	0.00	0.16	0.07
	50	0.23	0.10	0.39	0.12	**0.07**	-0.02	0.22	0.10
400	10	**0.00**	0.00	0.00	0.00	**0.00**	0.00	0.00	0.00
	20	0.10	0.06	0.16	0.04	**0.04**	-0.01	0.09	0.04
	30	0.16	0.08	0.27	0.09	**0.07**	0.03	0.13	0.04
	40	0.16	0.07	0.27	0.08	**0.04**	-0.02	0.09	0.04
	50	0.28	0.06	0.50	0.19	**0.08**	-0.01	0.19	0.08
450	10	**0.00**	0.00	0.00	0.00	**0.00**	0.00	0.00	0.00
	20	0.09	0.03	0.15	0.06	**0.02**	0.00	0.07	0.03
	30	0.17	0.05	0.28	0.10	**0.06**	-0.01	0.13	0.06
	40	0.19	0.06	0.38	0.13	**0.04**	-0.01	0.10	0.05
	50	0.24	0.07	0.37	0.12	**0.13**	0.02	0.22	0.08
500	10	**0.00**	0.00	0.00	0.00	**0.00**	0.00	0.00	0.00
	20	0.02	0.00	0.03	0.01	**0.01**	0.00	0.02	0.01
	30	0.12	0.04	0.23	0.08	**0.03**	-0.01	0.06	0.03
	40	0.20	0.05	0.34	0.12	**0.05**	0.01	0.08	0.03
	50	0.21	0.03	0.37	0.14	**0.03**	-0.03	0.12	0.06
Average		**0.25**	**0.10**	**0.42**	**0.13**	**0.09**	**0.02**	**0.18**	**0.06**

TABLE III. BEST RESULTS FOR THE BENCHMARKS

Instance	BS [16]	IG	GVNS-QL	Instance	BS [16]	IG	GVNS-QL	Instance	BS [16]	IG	GVNS-QL	Instance	BS [16]	IG	GVNS-QL
50-10-1	4127	4127	4127	150-40-1	15956	15999	**15955**	300-20-1	18837	**18836**	**18833**	400-50-1	37778	37832	**37774**
50-10-2	4283	4283	4283	150-40-2	18075	18140	**18074**	300-20-2	22032	22032	22032	400-50-2	38211	38213	**38210**
50-10-3	3262	3262	3262	150-40-3	16351	**16348**	**16347**	300-20-3	20229	**20228**	**20227**	400-50-3	37651	37715	**37641**
50-10-4	3217	**3216**	**3216**	150-40-4	14555	14580	**14533**	300-20-4	19483	19490	19483	400-50-4	40436	40450	40441
50-10-5	3470	3470	3470	150-40-5	17208	17225	17208	300-20-5	20705	20705	20705	400-50-5	35426	**35412**	**35418**
50-20-1	5646	5646	5646	150-50-1	20298	20347	20315	300-30-1	26487	26487	26487	450-10-1	23987	23987	23987
50-20-2	5814	5814	5814	150-50-2	19115	**19103**	**19104**	300-30-2	24260	24291	24260	450-10-2	26277	26277	26277
50-20-3	5793	5793	5793	150-50-3	19306	**19305**	**19301**	300-30-3	24363	24368	24368	450-10-3	25849	25849	25849
50-20-4	5795	5795	5799	150-50-4	20131	20181	20131	300-30-4	23705	23710	**23688**	450-10-4	26910	26910	26910
50-20-5	4869	4874	4869	150-50-5	19241	19270	**19235**	300-30-5	22544	22549	**22542**	450-10-5	25191	25191	25191
50-30-1	7223	**7220**	7235	200-10-1	12155	12155	12155	300-40-1	26572	26588	**26569**	450-20-1	27512	27539	27512
50-30-2	7330	7330	7337	200-10-2	12227	12227	12227	300-40-2	29158	29250	29166	450-20-2	27924	27924	27924
50-30-3	6844	6869	6868	200-10-3	12595	12595	12595	300-40-3	25261	25268	25268	450-20-3	28769	28779	28769
50-30-4	7571	7571	7578	200-10-4	12301	12301	12301	300-40-4	27438	27456	27442	450-20-4	28446	28446	28446
50-30-5	7333	7333	7333	200-10-5	12076	12076	12076	300-40-5	28760	28791	28771	450-20-5	28539	28539	28539
50-40-1	9130	9215	**9126**	200-20-1	14846	14864	14854	300-50-1	31522	31547	**31513**	450-30-1	35123	35153	35123
50-40-2	10094	10105	10107	200-20-2	14086	14086	14086	300-50-2	29357	**29349**	**29351**	450-30-2	32492	32492	**32477**
50-40-3	9765	9861	9767	200-20-3	16115	16115	16115	300-50-3	30634	30688	**30633**	450-30-3	31950	31977	31959
50-40-4	9495	9504	9498	200-20-4	15972	15972	15972	300-50-4	32202	**32193**	**32192**	450-30-4	33691	33722	33691
50-40-5	8904	8951	8904	200-20-5	14174	14178	**14170**	300-50-5	28965	29073	29006	450-30-5	33614	**33603**	**33610**
50-50-1	11064	11715	11533	200-30-1	17034	**17021**	**17031**	350-10-1	19297	19302	19300	450-40-1	39547	39547	**39535**
50-50-2	10857	10934	**10854**	200-30-2	16952	16984	17033	350-10-2	21316	21316	21317	450-40-2	35939	35958	**35937**
50-50-3	10841	10915	10847	200-30-3	17420	**17412**	**17416**	350-10-3	21330	21330	21330	450-40-3	37794	37807	**37792**
50-50-4	9858	9912	9899	200-30-4	19986	**19983**	**19981**	350-10-4	21759	21759	21759	450-40-4	37596	37668	**37594**
50-50-5	11316	11422	**11304**	200-30-5	17966	17968	**17962**	350-10-5	20591	20591	20591	450-40-5	35681	35681	**35680**
100-10-1	6570	6570	6575	200-40-1	19895	**19890**	**19889**	350-20-1	25413	25417	25413	450-50-1	37287	37311	**37274**
100-10-2	5798	5802	5798	200-40-2	21637	21682	21690	350-20-2	27185	27185	27185	450-50-2	43323	43358	43330
100-10-3	6533	6533	6533	200-40-3	20542	20605	20544	350-20-3	22880	22880	22880	450-50-3	44010	44073	44074
100-10-4	6158	6158	6158	200-40-4	17322	17394	**17305**	350-20-4	22968	22968	22968	450-50-4	41014	**41013**	**41006**
100-10-5	6654	6654	6654	200-40-5	21194	21213	21209	350-20-5	22746	22787	**22745**	450-50-5	40923	40944	**40922**
100-20-1	8606	8606	8606	200-50-1	22580	22699	**22579**	350-30-1	25192	**25184**	**25184**	500-10-1	28839	28839	28839
100-20-2	8217	8220	8217	200-50-2	23410	23430	**23406**	350-30-2	27739	27743	**27738**	500-10-2	27923	27923	27923
100-20-3	9043	9043	9043	200-50-3	22276	22276	**22271**	350-30-3	27638	27657	27638	500-10-3	27349	27349	27349
100-20-4	8970	8970	8970	200-50-4	23918	**23916**	**23904**	350-30-4	29295	29295	29295	500-10-4	27575	27575	27575
100-20-5	9109	9109	9109	200-50-5	24275	**24274**	**24274**	350-30-5	25209	25212	**25205**	500-10-5	27457	27457	27457
100-30-1	11200	11203	**11198**	250-10-1	16639	16639	16639	350-40-1	29020	29081	**29008**	500-20-1	35948	35948	35948
100-30-2	10938	**10934**	**10934**	250-10-2	15476	15476	15476	350-40-2	28950	**28946**	**28949**	500-20-2	34129	34129	34129
100-30-3	10523	10523	**10515**	250-10-3	14872	14872	14872	350-40-3	36247	36252	36247	500-20-3	31064	31064	31064
100-30-4	11089	**11086**	**11087**	250-10-4	15247	15247	15247	350-40-4	34644	34702	34659	500-20-4	30887	30887	**30883**
100-30-5	10983	10983	**10981**	250-10-5	15026	15026	15026	350-40-5	29742	29832	**29738**	500-20-5	33768	33776	33768
100-40-1	12551	12616	**12550**	250-20-1	17577	17577	17577	350-50-1	32065	32065	**32060**	500-30-1	36337	36349	**36327**
100-40-2	13117	13117	**13112**	250-20-2	17683	17683	17683	350-50-2	32760	32923	**32750**	500-30-2	39346	39365	39346
100-40-3	12411	12424	**12410**	250-20-3	17487	17487	**17485**	350-50-3	34682	**34674**	**34672**	500-30-3	39226	39251	39240
100-40-4	11680	11763	11681	250-20-4	17639	17646	17639	350-50-4	36957	36960	**36948**	500-30-4	33890	33900	**33875**
100-40-5	12877	12877	12877	250-20-5	17274	**17273**	**17273**	350-50-5	35343	35352	**35334**	500-30-5	38340	38356	38340
100-50-1	15998	**15994**	**15996**	250-30-1	21918	21920	21918	400-10-1	25238	25238	25238	500-40-1	40685	40721	40716
100-50-2	14761	14774	**14756**	250-30-2	21814	21838	21814	400-10-2	23001	23001	23001	500-40-2	44099	44131	44136
100-50-3	17514	17617	**17513**	250-30-3	20077	20080	20083	400-10-3	23665	23665	23665	500-40-3	40313	**40309**	**40296**
100-50-4	16569	16646	**16563**	250-30-4	19744	19787	19744	400-10-4	23275	23275	23275	500-40-4	41886	**41882**	**41868**
100-50-5	14746	14772	**14740**	250-30-5	20881	20891	20884	400-10-5	21956	21956	21956	500-40-5	36037	36075	**36026**
150-10-1	10404	10404	10404	250-40-1	22743	22770	**22740**	400-20-1	27686	27686	27686	500-50-1	46175	**46164**	**46154**
150-10-2	8824	8824	8824	250-40-2	24082	**24056**	**24076**	400-20-2	28088	28088	28088	500-50-2	43272	43301	**43267**
150-10-3	9180	9180	9180	250-40-3	24314	**24313**	24314	400-20-3	26224	26227	26224	500-50-3	45147	**45146**	**45134**
150-10-4	10032	10032	10032	250-40-4	24741	24768	24762	400-20-4	25105	25162	**25104**	500-50-4	42343	**42341**	**42334**
150-10-5	9866	9870	9870	250-40-5	23443	**23435**	23449	400-20-5	24675	24686	**24667**	500-50-5	43000	43046	**42991**
150-20-1	10758	**10757**	**10755**	250-50-1	28511	28529	**28507**	400-30-1	29405	29447	**29399**				
150-20-2	11696	11699	11696	250-50-2	24241	24271	24244	400-30-2	29180	29252	29235				
150-20-3	12046	12060	12046	250-50-3	26351	**26347**	**26345**	400-30-3	28633	**28632**	**28630**				
150-20-4	10887	10887	10887	250-50-4	25410	**25406**	**25407**	400-30-4	31276	31276	**31270**				
150-20-5	13210	13210	13210	250-50-5	27332	**27331**	**27327**	400-30-5	34533	34539	34533				
150-30-1	15497	**15496**	**15492**	300-10-1	17498	17498	17498	400-40-1	37426	37448	**37418**				
150-30-2	13667	13683	13667	300-10-2	17350	17350	17350	400-40-2	33805	33812	**33797**				
150-30-3	14650	14651	**14647**	300-10-3	18627	18627	18627	400-40-3	34450	34455	**34448**				
150-30-4	14544	14550	14548	300-10-4	16941	16941	16941	400-40-4	35245	35263	**35235**				
150-30-5	15245	15288	**15242**	300-10-5	17521	17524	17521	400-40-5	32727	32800	**32716**				

V. CONCLUSION

In this paper, a novel GVNS algorithm through Q-learning was presented for the NIFSP with the makespan criterion. In the outer loop, insertion and exchange operators are employed, while, in the inner loop of VND, VIG_{ALL} and VBIH algorithms are used. Effective insertion and RIS local searches are also employed in another VND loop, in the developed VIG_{ALL} and VBIH algorithms. The parameters are determined through a Q-learning approach in the proposed GVNS-QL algorithm, instead of using constant parameter values.

The developed GVNS-QL algorithm was compared with the widely known IG algorithm using the benchmark set of [11]. The results indicate that the GVNS-QL outperforms the traditional IG algorithm in terms of RPD values. Additionally, the results of IG and GVNS-QL were compared with the current best-known results provided by [16]. The results state that the developed GVNS-QL algorithm improves the current best-known results for 104 out of 250 benchmark instances.

In future research, other metaheuristic algorithms can be developed for the problem. Other performance measures can also be studied for the problem such as total completion time and total tardiness. Additionally, developed GVNS-QL can be employed to solve other scheduling problems.

REFERENCES

[1] M. R. Garey, D. S. Johnson, and R. Sethi "The complexity of flowshop and job shop scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117–129, 1976.

[2] P. Baptiste and K. H. Lee, "A branch and bound algorithm for the F|no-idle|Cmax," in *Proceedings of the International Conference on Industrial Engineering and Production Management (IEPM'1997)*, pp. 429–438, 1997.

[3] P. Vachajitpan, "Job sequencing with continuous machine operation," *Comput. Ind. Eng.*, vol. 6, no. 3, pp. 255–259, 1982.

[4] N. E. H. Saadani, P. Baptiste, and M. Moalla, "The simple F2//C max with forbidden tasks in first or last position: A problem more complex than it seems," *Eur. J. Oper. Res.*, vol. 161, no. 1, pp. 21–31, 2005.

[5] I. Adiri and D. Pohoryles, "Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times," *Nav. Res. Logist.* Q., vol. 29, no. 3, pp. 495–504, 1982.

[6] N. E. H. Saadani, A. Guinet, and M. Moalla, "A traveling salesman approach to solve the F/no−idle/Cmax problem," *Eur. J. Oper. Res.*, vol. 161, no. 1, p.p. 11–20, 2005.

[7] P. J. Kalczynski and J. Kamburowski, "A heuristic for minimizing the makespan in no-idle permutation flow shops," *Comput. Ind. Eng.*, vol. 49, no. 1, pp. 146–154, 2005.

[8] D. Baraz and G. Mosheiov, "A note on a greedy heuristic for flow-shop makespan minimization with no machine idle-time," *Eur. J. Oper. Res.*, vol. 184, no. 2, pp. 810–813, 2008.

[9] Q. K. Pan and L. Wang, "A novel differential evolution algorithm for no-idle permutation flow-shop scheduling problems," *Eur. J. Ind. Eng.*, vol. 2, no. 3, pp. 279–297, 2008.

[10] Q. K. Pan and L. Wang, "No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 39, no. 7–8, pp. 796–807, 2008.

[11] R. Ruiz, E. Vallada, and C. Fernandez-Martinez, "Scheduling in flowshops with no-idle machines," in U.K. Chakraborty (Ed.), *Computational Intelligence in Flow Shop and Job Shop Scheduling*, Springer Berlin Heidelberg, pp. 21-51, 2009.

[12] G. Deng and X. Gu, "A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion," *Comput. Oper. Res.*, vol. 39, no. 9, pp. 2152–2160, 2012.

[13] M. F. Tasgetiren, Q. K. Pan, P. N. Suganthan, and O. Buyukdagli, "A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem," *Comput. Oper. Res.*, vol. 40, no. 7, pp. 1729–1743, 2013.

[14] Y. Zhou, H. Chen, and G. Zhou, "Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem," *Neurocomputing*, vol. 137, pp. 285–292, 2014.

[15] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, pp. 278–285, 1993.

[16] W. Shao, D. Pi, and Z. Shao, "Memetic algorithm with node and edge histogram for no-idle flow shop scheduling problem to minimize the makespan criterion," *Appl. Soft Comput. J.*, vol. 54, pp. 164–182, 2017.

[17] M. F. Tasgetiren, Q. K. Pan, P. N. Suganthan, and A. Oner, "A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion," *Appl. Math. Model.*, vol. 37, no. 10–11, pp. 6758–6779, 2013.

[18] W. Shao, D. Pi, and Z. Shao, "A hybrid discrete teaching-learning based meta-heuristic for solving no-idle flow shop scheduling problem with total tardiness criterion," *Comput. Oper. Res.*, vol. 94, pp. 89–105, 2018.

[19] C. Ling-Fang, W. Ling, and W. Jing-Jing, "A two-stage memetic algorithm for distributed no-idle permutation flowshop scheduling problem," *Chinese Control Conf. CCC*, vol. 2018–July, pp. 2278–2283, 2018.

[20] F. L. Rossi and M. S. Nagano, "Heuristics for the mixed no-idle flowshop with sequence-dependent setup times and total flowtime criterion," *Expert Syst. Appl.*, vol. 125, pp. 40–54, 2019.

[21] N. Mladenovic and P. Hansen, "Variable neighborhood search," *Comput. Oper. Res.*, vol. 24, no. 11, pp. 1097–1100, 1997.

[22] P. Hansen, N. Mladenović, and D. Urošević, "Variable neighborhood search and local branching," *Comput. Oper. Res.*, vol. 33, no. 10, pp. 3034–3045, 2006.

[23] G. Kirlik and C. Oguz, "A variable neighborhood search for minimizing total weighted tardiness with sequence-dependent setup times on a single machine," *Comput. Oper. Res.*, vol. 39, no. 7, pp. 1506–1520, 2012.

[24] M. F. Tasgetiren, O. Buyukdagli, Q. K. Pan, and P. N. Suganthan, "A general variable neighborhood search algorithm for the no-idle permutation flowshop scheduling problem," in *International Conference on Swarm, Evolutionary, and Memetic Computing*, pp. 24–34, 2013.

[25] G. M. Komaki, S. Mobin, E. Teymourian, and S. Sheikh, "A general variable neighborhood search algorithm to minimize the makespan of the distributed permutation flowshop scheduling problem," *Int. Sch. Sci. Res. Innov.*, vol. 9, no. 8, pp. 2582–2589, 2015.

[26] M. Komaki and B. Malakooti, "General variable neighborhood search algorithm to minimize the makespan of the distributed no-wait flow shop scheduling problem," *Prod. Eng.*, vol. 11, no. 3, pp. 315–329, 2017.

[27] S.F. Rad, R. Ruiz, N. Boroojerdian, "New high performing heuristics for minimizing makespan in permutation flowshops", *Omega*, vol. 37, no. 2, p.p. 331–45, 2009.

[28] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, Jan. 1983.

[29] I. Osman, C. Potts, "Simulated annealing for permutation flow-shop scheduling," *Omega*, vol. 17 (6), p.p. 551–557, 1989.

[30] R. Ruiz , T. Stützle , "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem", *Eur. J. Oper. Res.*, vol. 177, p.p. 2033–2049, 2007.

[31] J. Dubois-Lacoste, F. Pagnozzi, and T. Stützle, "An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem," *Comput. Oper. Res.*, vol. 81, pp. 160–166, 2017.

[32] H. Öztop , M. F. Tasgetiren, D.T. Eliiyi and Q. K. Pan, "Metaheuristic algorithms for the hybrid flowshop scheduling problem," *Comput. Oper. Res.*, vol. 111, pp. 177–196, 2019.

[33] S.S. Choong, L-P. Wong, and C.P. Lim, "Automatic design of hyper-heuristic based on reinforcement learning," *Information Sciences*, vol. 436-437, pp. 89-107, 2018.

[34] H. Öztop, M. F. Tasgetiren, D.T. Eliiyi, and Q. K. Pan, "Iterated greedy algorithms for the hybrid flowshop scheduling with total flow time minimization," *in Proceedings of the Genetic and Evolutionary Computation Conference, ACM*, pp. 379–385, 2018. https://doi.org/10.1145/3205455.3205500.