Contents lists available at GrowingScience

Journal of Project Management

homepage: www.GrowingScience.com

Solving blocking flowshop scheduling problem with makespan criterion using q-learning-based iterated greedy algorithms

M. Fatih Tasgetiren[a], Damla Kizilay[b*] and Levent Kandiller[c]

[a]*Deceased, Dead August 27, 2023*
[b]*Industrial Engineering Department, Izmir Democracy University, Izmir, Turkey*
[c]*Industrial Engineering Department, Yasar University, Izmir, Turkey*

C H R O N I C L E

Article history:
Received: December 20, 2023
Received in revised format: January 2, 2024
Accepted: February 15, 2024
Available online:
February 15, 2024

Keywords:
Q-learning-based iterated greedy algorithms
Reinforcement learning
Blocking flowshop scheduling problem

A B S T R A C T

This study proposes Q-learning-based iterated greedy (IGQ) algorithms to solve the blocking flowshop scheduling problem with the makespan criterion. Q learning is a model-free machine intelligence technique, which is adapted into the traditional iterated greedy (IG) algorithm to determine its parameters, mainly, the destruction size and temperature scale factor, adaptively during the search process. Besides IGQ algorithms, two different mathematical modeling techniques. One of these techniques is the constraint programming (CP) model, which is known to work well with scheduling problems. The other technique is the mixed integer linear programming (MILP) model, which provides the mathematical definition of the problem. The introduction of these mathematical models supports the validation of IGQ algorithms and provides a comparison between different exact solution methodologies. To measure and compare the performance of IGQ algorithms and mathematical models, extensive computational experiments have been performed on both small and large VRF benchmarks available in the literature. Computational results and statistical analyses indicate that IGQ algorithms generate substantially better results when compared to non-learning IG algorithms.

1. Introduction

Johnson (1954) proposed a permutation flowshop scheduling problem (PFSP), which is the simplest version of the flowshop scheduling problems, for the first time in the literature. In PFSP, each job is processed on several machines. The route of each job is the same. Also, all the jobs have the same permutation on each machine. PFSP has a variety of applications in the industry (Blazewicz et al., 2007) and can be applied to sectors such as textile, plastic, chemical, and semiconductor (Pan & Ruiz, 2012). Also, PFSP has received significant attention from the literature over several years (Fernandez-Viagas et al., 2016, 2017) and was proved to be NP-hard when the objective is minimizing the makespan (Garey et al., 1976). For the last fifty years, various mathematical models have been developed for several extensions of the PFSP models to meet the needs of the industry (Cheng et al., 2019). In addition, a wide range of objective function values were considered for the PFSP (Birgin et al., 2020; Ramezanian et al., 2019). In PFSP, the buffer spaces between machines are assumed as unlimited. However, in some production plants, it cannot be possible to have unlimited buffer spaces (Miyata & Nagano, 2019). Therefore, a blocking permutation flowshop scheduling problem (BFSP) has arisen as a variant of PFSP. In BFSP, there is not any buffer space between the consecutive machines. Thus, jobs cannot move to the next machine if that machine is not available. Since there is no buffer area, jobs cannot move anywhere after they complete the process on the current machine, so have to stay at the current machine without being processed. At that time period, none of the jobs can be processed by the blocked current machine. When the upstreaming machine becomes available, then the job leaves the current machine

* Corresponding author.
E-mail address: damla.kizilay@idu.edu.tr (D. Kizilay)

doi: 10.5267/j.jpm.2024.2.002

and allows next job to be processed. These blocking situations can occur in several production types, such as robotic cells (Carlier et al., 2010; Ribas et al., 2015; Ribas & Companys, 2015), and modern industrial production systems (Shao et al., 2018a). Furthermore, several sectors have blocking constraints and actively apply BFSP, i.e., the chemical and pharmaceutical sectors (Merchan & Maravelias, 2016). In this sector, buffer areas are not allowed in production for health and hygiene reasons. Waiting in the buffer area causes the structures of chemical products to deteriorate and the effects of the drugs to disappear. BFSP is also applied in the iron and steel industry (Gong et al., 2010), in which the structure of the products waiting in the buffer areas is damaged due to oxidation. In addition, BFSP is suitable for the electronic manufacturing shop (Chen et al., 2014), where in some electronic products, waiting periods between production processes may damage the product structure.

Several studies provided mixed integer linear programming (MILP) model formulation to obtain optimal solutions for the BFSP (Ronconi & Armentano, 2001) and several variants of BFSP, such as distributed BFSP (Naderi & Ruiz, 2010) and mixed BFSP (Trabelsi et al., 2012). The BFSP can be solved optimally if the number of machines equals two (Gilmore et al., 1984). However, it becomes NP-Hard when the machine numbers are greater than two (Hall & Sriskandarajah, 1996). The problem's computational complexity causes exact algorithms to fail to handle large and medium-sized instances, so heuristic and meta-heuristics are generally employed to solve the BFSP and its variants. Besides the heuristics and metaheuristics, several constructive heuristic algorithms were developed by the authors, such as profile fitting (Mccormick et al., 1989) and NEH (Nawaz et al., 1983) heuristics. Also, many heuristic algorithms are built on profile fitting and NEH algorithms by modifying these algorithms (Ronconi & Armentano, 2001; Débora P Ronconi, 2004). An improving heuristic (Mccormick et al., 1989) and several NEH-based constructive heuristics were proposed (Newton et al., 2019; Riahi et al., 2017, 2019). These constructive heuristics provide a good initial solution to the heuristic or meta-heuristic algorithms. These metaheuristics are, i.e., genetic algorithms (Caraffa et al., 2001), several variants of artificial bee colony algorithms (Han et al., 2012, 2013, 2015), harmony search (Wang, Pan, & Tasgetiren, 2010), tabu search (Glover, 1990), iterated greedy (Ribas et al., 2019; Tasgetiren et al., 2017), variable block insertion (Tasgetiren et al., 2016), differential evolution (Wang, Pan, Suganthan, et al., 2010), particle swarm (Wang & Tang, 2012), fruit fly optimization (Han et al., 2016), water wave optimization (Shao et al., 2018b, 2019) algorithms and so on.

Moreover, some studies address the specific needs of the industry such as sequence-dependent setup times integrated into the BFSP by Shao, et al. (2020). Also, time constraints for BSFP are considered by Chen et al. (2014). Multi-objective optimization of energy-efficient BFSP is considered by Kizilay et al. (2019), and very recently, Han et al. (2020) integrated a setup time to similar considerations. A BFSP group scheduling problem integrated with the transfer times is considered by Yuan et al. (2020). A hybrid BFSP is handled by the two studies (Aqil & Allali, 2021; Elmi & Topaloglu, 2013), while parallel BFSP is also considered in the literature (Ribas et al., 2017, 2019). Furthermore, a lot-streaming BFSP is presented by Gong et al. (2018). In recent years, a distributed BFSP with makespan has been considered by Zhang et al. (2018) and solved using discrete differential evolution algorithms, while Shao, et al. (2020) consider a fuzzy distributed BFSP. A detailed review of BFSP literature is provided (Miyata & Nagano, 2019).

Very recently, multiple perturbation operators were incorporated into their iterated greedy (IG) algorithm, denoted as QIG. They employed the Q-learning approach, one of the machine learning techniques, to select the perturbation strength of the destruction and construction operator for the PFSP. They show that QIG outperforms even the algorithms that achieve the best results in the literature for scheduling problems to date. Note that a similar Q-learning is employed to solve the no-idle PFSP (Öztop et al., 2020; Öztop et al., 2022) as mentioned by Karimi-Mamaghan et al. (2022). In this study, we utilize the Q-learning approach to solve the BFSP and develop our Q-learning-based IG algorithms, denoted as IGQ1 and IGQ2, to compare to IG, IGALL, and QIG algorithms. Computational results indicate that IG algorithms with Q-learning, namely, IGQ1, IGQ2, and QIG, substantially outperform the traditional IG algorithm.

The following is explained in the upcoming sections of the article. In section 2, the CP and MILP mathematical models proposed for problem-solving are explained and their formulations are given. In section 3, IG and IGALL algorithms, which are traditional metaheuristic approaches in the literature, are explained. Additionally, the local search (LS) procedure used in these algorithms is also explained. Sections 4 and 5 summarize reinforcement learning (RL) and Q-learning approaches. Comparative results and performances of all developed and proposed models and algorithms are explained in section 6. In the last section, section 7, the results obtained are summarized and information about future studies is given.

2. Mathematical Formulations

BFSP is formulated using MILP and CP models. Both models use the same parameters, which are presented in the following. We have several jobs and machines, the process duration time of each job, and a sufficiently big integer used in only the CP model. We used the MILP model formulation of Ronconi & Armentano (2001) and developed a CP model. Both models, including their objectives and constraints are explained in the following parts.

Parameters:

N:	Job set $\{1, \dots, n\}$
M:	Machine set $\{1, \dots, m\}$
$ptime_{i,j}$:	Process duration of job $i \in N$ on machine $j \in M$

D: A sufficiently big integer value

2.1. Mixed-Integer Linear Programming Model

The MILP model is constructed by introducing specific decision variables, an objective function, and various constraints. The decision variables consist of two types of integers representing the job finish time on individual machines and the maximum finish time, respectively. Additionally, a binary variable is utilized to denote the positions of the jobs on the machines, ensuring that job permutations remain consistent across all machines. The model is designed to optimize the job schedules on the machines, considering the completion times and processing orders, with the ultimate goal of achieving efficient and effective scheduling outcomes.

Decision variables:

$C_{i,j}$: The end time of a process of job $i \in N$ on machine $j \in M$

$x_{i,j} = \begin{cases} 1, & \text{if job } i \text{ is processed at position } j \text{ on the machines} \\ 0, & \text{otherwise} \end{cases}$

$Cmax$: The maximum of the process end time of the jobs

Objective Function

$$Minimize\ Cmax \tag{1}$$

Constraints

$$Cmax \geq C_{i,m} \qquad \forall i \in N \tag{2}$$

$$C_{i,1} \geq \sum_{i \in N} \left(x_{i,j} * ptime_{i,1}\right) \qquad \forall j \in N \tag{3}$$

$$C_{j,k} \geq C_{j,k-1} + \sum_{i \in N} \left(x_{i,j} * ptime_{i,k}\right) \qquad \forall j \in N, k \in M | k \geq 2 \tag{4}$$

$$C_{j,k} \geq C_{j-1,k} + \sum_{i \in N} \left(x_{i,j} * ptime_{i,k}\right) \qquad \forall j \in N | j \geq 2, k \in M \tag{5}$$

$$C_{j,k} \geq C_{j-1,k+1} \qquad \forall j \in N | j \geq 2, k \in M | k \leq m-1 \tag{6}$$

$$\sum_{i \in N} x_{i,j} = 1 \qquad \forall j \in N \tag{7}$$

$$\sum_{j \in N} x_{i,j} = 1 \qquad \forall i \in N \tag{8}$$

This study's objective (1) is to achieve the lowest makespan value, which can be calculated as the last processed job's production end time on the last machine, as stated in constraint (2). Constraints (3-5) calculate the jobs' completion time regarding the problem's blocking variant. Constraint (6) states that a job is processed through a series of machines. Assignment constraints (7-8) provide that each job is fixed in a single position in the sequence and each position has a job at each machine.

2.2. Constraint Programming Model

The CP model incorporates both interval and sequence variables, which are the expressions that make it easier to build scheduling models. Interval variables represent the start time of the process, duration of the process, and end time of each job's process on the machines. Specifically, we define interval variables for each job and machine to track the temporal characteristics of their processing. On the other hand, sequence variables are established for each machine, detailing the set of interval variables associated with the respective machines. This approach allows us to model and optimize the sequencing of jobs on the machines effectively, considering both their temporal properties and the overall operations on each machine.

Decision variables

$y_{i,k}$: Interval variable denotes processing of job $i \in N$ on machine $k \in M$ with a duration between $ptime_{i,j}$ and D.

z_k: Sequence variable of machine k defined over interval variables $y_{i,k}$.

Objective Function

$$minimize\ \max_{i \in N} endOf\left(y_{i,m}\right) \tag{9}$$

Constraints

$$endAtStart(y_{i,k}, y_{i,k+1}) \quad \forall i \in N, k \in M | k \leq m-1 \quad (10)$$

$$noOverlap(z_k) \quad \forall k \in M \quad (11)$$

$$sameSequence(z_1, z_k) \quad \forall k \in M | k > 1 \quad (12)$$

As seen from the equations the CP model is compact when compared to the MILP model. Since the CP has interval and sequence variables, the formulation of the scheduling problems is very easy and understandable. The objective function aims to obtain the minimum makespan value, which is achieved from the interval variable. Since it can be reached the end time point of an interval variable, minimizing the maximum end time of each job only for the last machine, provides the objective value (9). To enforce the blocking constraints that each job should start its processing on a machine immediately after it has completed processing on the previous machine, equation (10) is introduced. To ensure proper blocking between machines, an interval variable y is defined between the job processing times and a suitably large constant D. Constraint (11) states that an engine can operate only one job in the same period of time. This constraint is written with the help of the glocal constraint of noOverlap. The *noOverlap* constraint prevents interval variables from starting and ending within the same time period, ensuring that jobs on a machine are not executed at the same time. Furthermore, constraint (12) again uses the global constraint, which ensures that all jobs are operated in the same rank on all machines, in short, it is the constraint that ensures the permutation is the same.

3. Iterated Greedy Algorithms

In this section, the traditional IG algorithm first presented by Ruiz and Stützle (2007b) is explained. There are four important parts at the core of the IG algorithm. These parts are the initial solution, destruct-construct procedure, LS, and acceptance criteria. How these four parts are addressed is important. In this article, the NEH algorithm (Nawaz et al., 1983), which is a well-known algorithm in the literature is used for the initial solution. After obtaining the initial solution, the destruction part randomly removes dS number of jobs from the job list π obtained from the initial solution. Then, these removed, and the leftover jobs are kept in separate lists, π_d and π_r, respectively. The order of the remaining jobs constitutes the partial solution. LS has been applied to this partial solution. In the traditional IG structure, applying LS in this section is not mandatory, it is optional (Dubois-Lacoste et al., 2017). During the construction phase, previously extracted jobs were added to the job list obtained because of LS, one by one, in random positions in the order in which they were removed. After all the extracted jobs are added, the job list is completed, and the complete solution is obtained. Then, LS was applied again to the obtained solution. The applied LS procedure is explained in Algorithm 1. In this procedure, the insertion LS with speed-up technique, previously presented by Tasgetiren et al. (2017), and inspired by the speed-up techniques developed by Taillard (1990), was used. The purpose of applying this LS procedure is to quickly upgrade the solution quality by getting benefits from the speed-up techniques. If the current solution improves, the LS continues to be applied over the improved solution, otherwise, over the current solution. After the LS phase is completed, if the solution obtained is better than the in-process solution, it is saved as the new current solution; if not, it can be saved by checking it according to the (SA) acceptance criteria. At this point, a probability value for the acceptance rate is calculated respecting the objective function value and the temperature value T (Osman & N. Potts, 1989):

$$T = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} ptime_{ij}}{10nm} \times \tau P \quad (13)$$

where τP is a parameter to be determined. Finally, Algorithm 2 shows the pseudocode of the traditional IG algorithm implemented in this work.

Algorithm 1. LocalSearch(π^1)

for j=1 to n do
  π_k:= Randomly choose a job from π^1
  π^2:= Insert Job π_k in the best position of π^1
  if $\left(f(\pi^2)<f(\pi^1)\right)$ **then do**
    π^1:=π^2
  endif
endfor
return π^1 and $f(\pi^1)$

Algorithm 2. Traditional IG algorithms

```
π:=NEH
π:=InsertionLocalSearch(π)
π^best:=π
while (NotTermination) do
  π_d,π_r:=Destruction(π,dS)
  π_r:=InsertionLocalSearch(π_r)      //IGALL algorithm, optional
  π^1:=Construction(π_d,π_r)
  π^2:=InsertionLocalSearch(π^1)
  if (f(π^2)<f(π) ) then do
      π:=π^2
      if ( f(π^2)<f(π^best)) then do
          π^best:=π^2
      endif
  else if (rand()<exp{-(f(π^2)-f(π))/T}) then do
          π:=π^2
  endif
end while
return π^best and f(π^best)
```

4. Reinforcement Learning and Q-learning

RL technique is based on machine learning and its basis is to reward good behavior and punish bad behavior. An RL agent can perceive and interpret its environment, take action, and learn through trial and error to achieve a specific goal (Kaelbling et al., 1996; Sutton & Barto, 2018). The aim here is to enable learning by trial-and-error method by creating interaction with the environment. As the environment is learned, the reward obtained will also reach its maximum level. Most known RL methods require a model that includes all possible states, actions available for each state, transition probabilities between states, and expected rewards. However, often a complete model may not be available, or it may take a long time to obtain the complete model. For such situations, a model-independent RL algorithm called Q-learning has been developed (Watkins, 1989). The developed Q-learning algorithm is based on gradual differences. In this algorithm, there is the state space (S), the action space (A), the state-action pair (s, a) and the expected gain score $Q(s,a)$ obtained as a result of the action chosen for each situation. $Q(s,a)$ is calculated as follows:

$$Q(s, a)=Q(s, a)+\alpha[r+\gamma \max_{a'} Q(s',a') -Q(s,a)] \quad (14)$$

In this equation, s indicates the current state and a indicates the action applied in state s. Additionally, the next situation (s') and the action to be applied in the next situation (a') are also included in the equation. While the resulting score is updated, a section multiplied by the learning rate α ($0 < \alpha \leq 1$), is added to the existing score. In this section, the difference between the maximum of future scores and the current score is multiplied by the discount factor γ ($0 < \gamma \leq 1$), and the reward (r) for choosing action a is added.

In choosing an action for the current situation, both exploration and exploitation actions are important. At this point, while the Q-learning technique provides a balance between them, it also allows using state-action pairs that have never been discovered before. The ϵ-greedy strategy defined below is implemented using certain probability values (Sutton & Barto, 2018).

$$a=\begin{cases} \text{argmax}_{a\in A} Q(s, a) & \text{having } 1-\epsilon \text{ prob. value} \\ \text{any action with equal choice in A and chosen at random} & \text{having } \epsilon \text{ prob. value} \end{cases}$$

5. IGQ Algorithms with Q-learning

In this article, the Q-learning algorithm and the IG algorithm explained in the previous sections are used together in the IGQ algorithms. The learning mechanism in the RL and Q-learning algorithms was used to determine the parameter values of the IG algorithm self-adaptively in the IGQ algorithm. The Q value in the Q-learning function is calculated for each parameter-action pair in the IGQ algorithm. In this case, Eq. (15) was created by using the parameter (p) value instead of the state (s) value in equation 14. Ee define $Q(p,a)$, a function that determines both the temperature scale factor (τP) and the destruction size (dS) for the IGQ algorithms as follows:

$$Q(p, a)=Q(p, a)+\alpha[r+\gamma \max_{a'} Q(p', a') -Q(p, a)] \quad (15)$$

In the IGQ algorithm used, the parameter set includes temperature scale factor and destruction size values. The reward (r) is defined as $1/C_{max}$ since the objective function of the problem try to obtain the minimum C_{max} value and a smaller C_{max} value should lead to larger reward values. Moreover, if the target value of the obtained solution becomes worse than the current value during the iterations, it is still accepted with the SA-type acceptance criterion. At this point, a lower reward

should be achieved for the action performed, because a higher C_{max} value than the current value has been achieved. In the IGQ algorithm, clusters with different values for τP and dS were determined as A_{dS} and $A_{\tau\mathrm{P}}$. A set is defined as $A_{\tau\mathrm{P}} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and $A_{\mathrm{dS}} \in \{1, 2, 3\}$. It is important to bear in mind that in the IGQ1 algorithm, we employ the $A_{\mathrm{dS}} \in \{2, 3, 4, 5, 6, 7\}$, whereas the $A_{\mathrm{dS}} \in \{1, 2, 3\}$ is taken as an action list in the IGQ2 algorithm. In both algorithms, $A_{\tau\mathrm{P}}$ is taken as $A_{\tau\mathrm{P}} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. Other parameters are taken as $\epsilon := 0.8$, $\beta := 0.996$, $\alpha := 0.6, and\ \gamma := 0.8$, which are experimentally determined by Karimi-Mamaghan et al. (2022) and we barrow those values for our IGQ algorithms.

The proposed IGQ algorithms are almost the same as the traditional IG algorithm. However, we determine parameter set $p \in (\tau\mathrm{P}, \mathrm{dS})$ by using Q-learning algorithms at each iteration. In addition, we employ the referenced local search (RLS) (M. Fatih Tasgetiren et al., 2009), given in Algorithm 3, with the speed-up methods by Tasgetiren et al. (M. Fatih Tasgetiren et al., 2017) in Q-learning-based IGQ and QIG algorithms. Finally, we can outline IGQ algorithms in Algorithm 4.

Algorithm 3 **RLS**$(\boldsymbol{\pi}, \boldsymbol{\pi}^{\mathbf{best}})$

```
Cnt := 1, pos := 1
π^ref := π^best
while(Cnt ≤ n) do
        k ← 1
        while (π_k ≠ π^ref_pos) k := k + 1; break //Find job π_k from referenced permutation π^ref_pos
        pos := (pos + 1) % n                    // Repeat it until π_k is found
        π_p := remove π_k from π
        π* := Place the job π_k to the best − performing place in the sequence of π_p
        if (f(π*) < f(π)) then do
                π := π*
                Cnt := 1
        else
                Cnt := Cnt + 1
        endif
endwhile
return π and f(π)
```

Algorithm 4. IGQ algorithms

```
π := NEH
π := InsertionLocalSearch(π)
π^best := π
ϵ := 0.8; β := 0.996, α = 0.6, and γ = 0.8
τP =: RandomChoice(A_τP)
dS := RandomChoice(A_dS)
while (NotTermination) do
    π_d, π_r := Destruction(π, dS)
        π_r := InsertionLocalSearch(π_r)      //IGALL algorithm, optional
      π^1 := Construction(π_d, π_r)
      π^2 := RLS(π^1)
      if (f(π^2) < f(π)) then do
            r := 1.0/ f(π^2)
            π := π^2
            Q(p,a) = Q(p,a) + α[r + γ max_a' Q(p',a') − Q(p,a)]
            if (f(π^2) < f(π^best)) then do
                π^best := π^2
            endif
      else if
                T := (Σ_{i=1}^{n} Σ_{j=1}^{m} ptime_ij / (10. n. m)) × τP
                if (rand() < exp{−(f(π^2) − f(π))/T}) then do
                  π := π^2
                    Q(p,a) = Q(p,a) + α[r + γ max_a' Q(p',a') − Q(p,a)]
        endif
      endif
      ϵ := ϵ. β
      if (rand ≥ ϵ) then do
                τP := argmax Q(p''_τP, A''_τP)   //Next action is determined from Q-Table
                dS := argmax Q(p''_dS, A''_dS)   //Next action is determined from Q-Table
      else
                τP := RandomChoice(A''_τP)   //Next action is determined randomly
                dS := RandomChoice(A''_dS)   //Next action is determined randomly

      endif
endwhile
return π^best and f(π^best)
```

6. Computational Results

This study handles the BFSP problem and presents the mathematical formulation of the BFSP by using the MILP and CP models. In addition, several heuristic algorithms were developed to acquire good and qualified solutions in a short computational time. All the models and the heuristics were performed on small VRF instances, which are well-known benchmarks, proposed by Vallada et al. (2015). Both MILP and CP models were coded on OPL CPLEX Studio IDE 12.10 and given a 1-hour time limit to solve them to optimality or an upper bound with a GAP from optimality. Once we obtained results both from the MILP and CP, we chose the better ones amongst them as lower bounds for the small-sized VRF instances. Microsoft Visual Studio platform and C++ coding language were used to acquire the solutions for the developed metaheuristic algorithms. The results of all heuristic algorithms used were obtained by running these algorithms for 25*n*m milliseconds with five replications. The results of the mathematical model and heuristic algorithms were obtained using an Intel (R) Core (TM) i7-2600 desktop with a 3.40 GHz CPU and 8GB RAM. All the best-known solutions (BKS) for both small and large-size VRF instances (ins.) are given in Appendix A1 and Appendix A2.

6.1. Comparisons on the Small VRF Instances

In this section, the results of the CP model MILP model, and the proposed and developed heuristic algorithms (IG, IGALL, IGQ1, IGQ2, and QIG) are compared considering the relative deviations from the obtained results for the small VRF examples. The average relative percent deviations (ARPD) of the mathematical models are calculated by the following equation:

$$RPD = \frac{f(x)-f(best)}{f(best)} * 100 \tag{16}$$

In Eq.16, f(x) is the obtained solution from the models, and f(best) is the optimal or the best solution of the models, MILP and CP. Therefore, relative percent deviations (RPD) of the algorithms from the best or optimal results obtained from the models are calculated. Regarding the metaheuristics, we also provide the relative percent improvements as:

$$RPI = \frac{f(best) - UB}{UB} * 100 \tag{17}$$

In Eq. (17), upper bounds (UB) are taken either from MILP or CP results, and f(best) is the obtained solution of IG, IGALL, IGQ1, QIG, and IGQ2 algorithms. In VRF small instances, there are six different job sizes (10, 20, 30, 40, 50, 60) and four different machine numbers (5, 10, 15, 20). There exist ten instances for each $job \times machine$ combination. The calculated RPD and RPI values of the instances are gathered, and the average for each $job \times machine$ combination (average of ten instances) is calculated as the average relative percent deviation (ARPD) and average relative percent improvement (ARPI). Results of the heuristics were obtained by running them for $25\times n\times m$ milliseconds with five replications for each combination. For each $n \times m$ combination, Table 1 presents the ARPI values of the average solutions of these five replications, as well as the ARPD values of MILP and CP models.

Table 1
ARPD of the results for small VRF instances

		MILP			CP			ARPI				
n	**m**	**ARPD**	**CPU**	**GAP**	**ARPD**	**CPU**	**GAP**	**IG**	**IGALL**	**IGQ1**	**QIG**	**IGQ2**
10	5	0	0.57	0	0	20.11	0	0.02	0	0	0	0.01
10	10	0	1.01	0	0	138.21	0	0	0	0	0	0
10	15	0	3.71	0	0	331.84	0	0	0	0	0	0
10	20	0	7.11	0	0	663.51	0	0	0	0	0	0
20	5	0.08	3600	0.06	0.40	3600	0.12	-0.58	-0.61	-0.61	-0.61	-0.61
20	10	0.30	3600	0.10	0.48	3600	0.18	-0.35	-0.35	-0.36	-0.36	-0.36
20	15	0.79	3600	0.12	0.04	3600	0.20	-0.50	-0.51	-0.51	-0.51	-0.51
20	20	0.57	3600	0.16	0.11	3600	0.22	-0.50	-0.51	-0.51	-0.51	-0.51
30	5	0.95	3600	0.11	0.05	3600	0.12	-1.36	-1.45	-1.42	-1.46	-1.47
30	10	1.85	3600	0.19	0.01	3600	0.20	-1.59	-1.70	-1.67	-1.71	-1.71
30	15	2.35	3600	0.21	0.00	3600	0.21	-1.46	-1.62	-1.59	-1.61	-1.62
30	20	2.70	3600	0.23	0.00	3600	0.23	-1.79	-1.85	-1.82	-1.84	-1.84
40	5	1.97	3600	0.14	0.02	3600	0.13	-1.98	-2.22	-2.12	-2.19	-2.28
40	10	4.47	3600	0.24	0.00	3600	0.21	-2.05	-2.30	-2.20	-2.30	-2.33
40	15	4.19	3600	0.25	0.00	3600	0.22	-2.49	-2.69	-2.62	-2.68	-2.69
40	20	5.21	3600	0.27	0.00	3600	0.24	-2.16	-2.60	-2.52	-2.58	-2.60
50	5	3.07	3600	0.14	0.00	3600	0.12	-2.17	-2.35	-2.30	-2.31	-2.40
50	10	5.61	3600	0.24	0.00	3600	0.19	-2.12	-2.51	-2.36	-2.46	-2.52
50	15	4.43	3600	0.28	0.00	3600	0.24	-3.46	-3.87	-3.71	-3.84	-3.88
50	20	6.05	3600	0.29	0.00	3600	0.25	-2.35	-2.70	-2.54	-2.67	-2.68
60	5	4.20	3600	0.16	0.00	3600	0.13	-2.05	-2.30	-2.17	-2.27	-2.39
60	10	5.16	3600	0.25	0.00	3600	0.21	-3.00	-3.38	-3.16	-3.32	-3.41
60	15	6.10	3600	0.29	0.00	3600	0.24	-2.90	-3.47	-3.20	-3.43	-3.48
60	20	14.52	3600	0.34	0.00	3600	0.25	-3.33	-3.66	-3.49	-3.61	-3.64
Avg		**3.11**	**3000.52**	**0.17**	**0.05**	**3048.07**	**0.16**	**-1.59**	**-1.78**	**-1.70**	**-1.76**	**-1.79**

The overall average values are written in bold. The CPU represents the average computation time of the mathematical models, i.e., MILP and CP, respectively. The GAP value represents the difference between the lower and upper limits obtained because of solving the MILP and CP models within a limited time. The solution value obtained for the minimization problem is the upper limit value. The GAP value is obtained by finding the difference between this value and the lower limit and dividing it by the upper limit value. The GAP values written in the table were calculated by the solver. If the results are optimal, then the GAP values will be zero. Table 1 displays that, as the job size increases, the ARPD% values also increase for the models and the heuristics. However, the same comment cannot be made about the number of machines. The ARPD% values do not follow a smooth pattern. Regarding the overall average values, the best-performing algorithm is the IGQ2 with -1.79 ARPD%. Then IGALL and QIG follow with very small differences in their ARPD% values. All the heuristics have relatively small differences in their ARPD% values, but it is obvious that the performances of the models are worse. When the models were compared, the CP model generated better results than the MILP model respecting the solution quality. Although the models were given 3600 seconds, they could not find good results due to the computational complexity of the problem, especially for the MILP model. Thus, the MILP model is not employed for large-size instances. To see all the numbers, Table 2 summarizes the results of all models and algorithms, indicating the number of optimal solutions, best solutions, ARPD/ARPI, and average CPU times. The best solutions for each instance obtained by the models or the algorithms are recorded. Table 2 shows which model and algorithm found the best solution in how many of the 240 instances were presented as "# of best".

Table 2
Summary of the results for small VRF instances

	# of proven optimal	# of best	ARPD/ARPI	Avg. CPU (s)
MILP	40	41	3.11	3000.52
CP	40	43	0.05	3048.07
IG	39	80	-1.59	10.94
IGALL	40	149	-1.78	10.94
IGQ1	40	97	-1.70	10.94
QIG	40	127	-1.76	10.94
IGQ2	39	171	-1.79	10.94

Both MILP and CP models obtained the optimal solutions for all replications of the ten job instance sets. It corresponds to 40 instances out of 240 instances. However, if the job size reaches 20, models cannot find the optimal results within 3600 seconds and provide sub-optimal solutions. When the results for the ten jobs were investigated, it was seen that all the heuristic algorithms could find all the optimal results except for the IGQ2 and IG algorithms. However, these two algorithms cannot find optimal solutions only for one instance, and there is a very small difference. Most of the best results are found by the IGQ2 algorithm. Then the IGALL and QIG algorithms follow. However, the ARPI values of all the algorithms are very close to each other, so an interval plot is provided in Fig. 1 to show whether these algorithms' results are statistically different from each other. Fig. 1 compares the mean of the ARPI values of the algorithms under a 95% confidence interval.

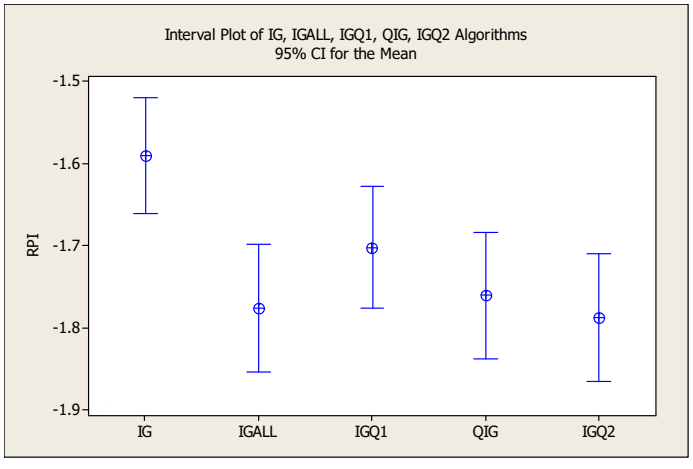

Fig. 1. Interval plot of IG, IGALL, IGQ1, QIG, and IGQ2 algorithms for small VRF instances

As seen from Fig. 1, all the algorithms, except IG, intersect each other, so they are not statistically different from each other. The IG algorithm does not intersect with the IGALL, QIG, and IGQ2 algorithms, so it is statistically significant that the results of the IG are worse than these algorithms. However, the IG and IGQ1 results are not significantly different.

6.2. Comparisons on the Large VRF Instances

This section provides a comparison between different algorithms on the same instance sets. Large-size VRF instances were used for the comparison. IG, IGALL, IGQ1, QIG, and IGQ2 algorithms were compared to each other. VRF large instances

include eight different job sizes (100, 200, 300, 400, 500, 600, 700, 800) and three different machine numbers (20, 40, 60). Each job and machine combination includes ten different instances. CP model was run for 1 hour to solve large data sets. Since the CP cannot provide good solutions under the given time limit, the results of the model are accepted as upper bound (UB). The deviations of the results of all algorithms from the CP model's results (upper bound) were calculated. Each algorithm was run in five iterations for each $job\ x\ machine$ combination. Table 3 shows the ARPI of the average of these replications of all algorithms.

Table 3
ARPI of the results for large VRF instances

n	m	IG	IGALL	IGQ1	QIG	IGQ2
100	20	-3.49	-3.90	-4.10	-4.12	-4.18
100	40	-3.53	-3.71	-3.99	-4.09	-4.03
100	60	-3.58	-3.72	-3.93	-4.01	-4.03
200	20	-5.41	-6.04	-6.29	-6.39	-6.39
200	40	-5.49	-6.01	-6.12	-6.29	-6.31
200	60	-6.25	-6.67	-6.79	-6.88	-6.87
300	20	-6.58	-7.42	-7.65	-7.82	-7.78
300	40	-6.99	-7.59	-7.72	-7.80	-7.84
300	60	-7.61	-8.03	-8.12	-8.22	-8.23
400	20	-7.80	-8.67	-8.86	-8.97	-9.02
400	40	-7.91	-8.54	-8.65	-8.78	-8.78
400	60	-8.01	-8.51	-8.60	-8.73	-8.71
500	20	-8.82	-9.70	-9.94	-10.08	-10.06
500	40	-8.39	-9.00	-9.16	-9.29	-9.32
500	60	-8.14	-8.66	-8.78	-8.86	-8.85
600	20	-9.94	-10.91	-11.04	-11.27	-11.25
600	40	-8.81	-9.45	-9.57	-9.70	-9.76
600	60	-8.21	-8.76	-8.85	-8.98	-8.96
700	20	-10.66	-11.57	-11.72	-11.96	-11.89
700	40	-8.72	-9.38	-9.52	-9.75	-9.68
700	60	-8.41	-8.94	-9.04	-9.21	-9.12
800	20	-10.73	-11.68	-11.82	-12.07	-12.04
800	40	-8.91	-9.58	-9.71	-9.89	-9.84
800	60	-8.48	-9.02	-9.07	-9.25	-9.25
Avg.		**-7.54**	**-8.15**	**-8.29**	**-8.43**	**-8.42**

As seen in Table 3, all five algorithms provide similar results with small differences in their overall ARPI values. The best-performing algorithms are the QIG and IGQ2 algorithms, with -8.43 and -8.42 ARPI values, respectively. As the number of jobs increases, the improvement performance of algorithms increases. This is because the CP model achieves worse results within 1 hour as the number of jobs increases. While the number of jobs is 500 and greater than 500, the improvement percentages of the algorithms decrease as the number of machines increases. However, the same trend is not valid for the number of jobs less than 500; no significant decrease or increase was observed in these data sets according to the number of machines. On the other hand, it is obvious that learning-based algorithms perform the best.

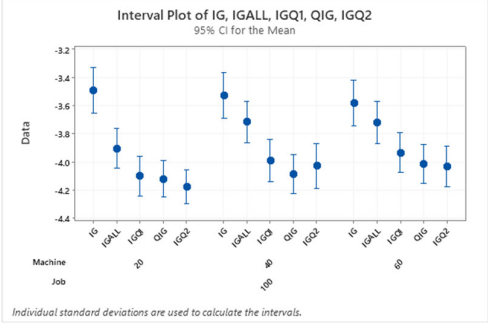

Fig. 2. Interval plot for 100 jobs and 20, 40, and 60 machines

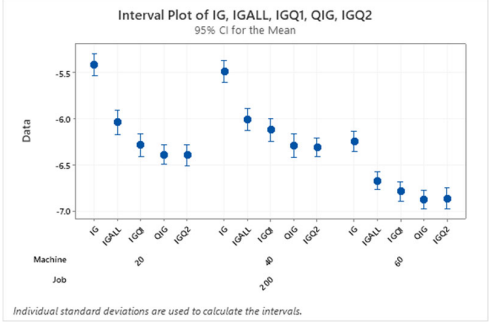

Fig. 3. Interval plot for 200 jobs and 20, 40, and 60 machines

Fig. 2 plots the range graph of the meta-heuristic algorithms for 20, 40, and 60 machines and 100 jobs. A 95% confidence interval is assumed in this graph. According to the graph, the differences in RPIs become statistically significant as long as the confidence intervals of the two selected algorithms do not overlap. For each machine size, all metaheuristics follow a similar pattern for 20, 40, and 60 machines, as seen in Fig. 2. For 20 machines, the confidence intervals of IGQ1, QIG, and IGQ2 algorithms do not intersect with the IG algorithm's confidence intervals, and they have a small intersection with the IGALL algorithm; hence, their differences are statistically significant when compared to the traditional IG algorithm as well as the IGALL. Even the IGALL algorithm is statistically significant to the traditional IG algorithm. For 40 machines,

the results from IGQ1, QIG, and IGQ2 algorithms are statistically significant when compared to the IG and IGALL algorithms since their confidence intervals do not coincide. Similarly, for 60 machines, a similar pattern can be observed. Ultimately, it can be concluded that the proposed IG algorithms with Q-Learning outperform the traditional IG algorithm.

Fig. 3 presents the interval plot of metaheuristic algorithms for the 200 jobs having 20, 40, and 60 machines under the 95% confidence interval. For each machine size, all metaheuristics follow a similar pattern for 20, 40, and 60 machines, as seen in Fig. 2, too. For each machine combination, the confidence intervals of IGALL, IGQ1, QIG, and IGQ2 algorithms do not intersect with the IG algorithm's confidence intervals, so their differences are statistically significant compared to the traditional IG. Since the IG algorithms with QL generate better results than IGALL algorithms, they have a small intersection between their confidence intervals. Ultimately, it can be concluded that the proposed IG algorithms with QL outperform the traditional IG algorithm.

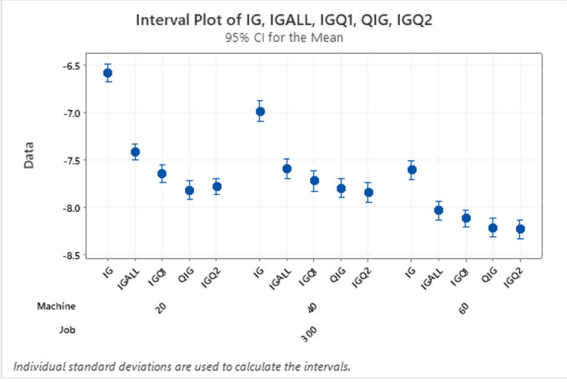

Fig. 4. Interval plot for 300 jobs and 20, 40, and 60 machines

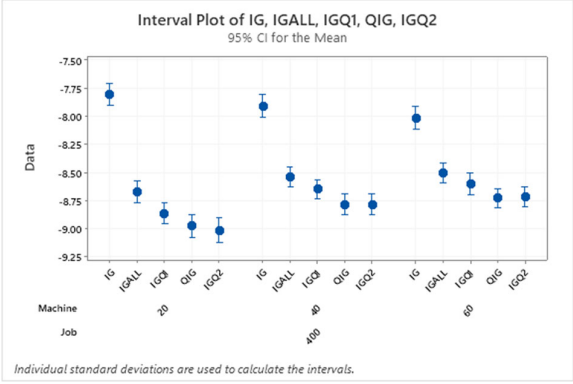

Fig. 5. Interval plot for 400 jobs and 20, 40, and 60 machines

Fig. 4 and 5 present the interval plot of metaheuristic algorithms for the 300 and 400 jobs, respectively, with 20, 40, and 60 machines under the 95% confidence interval. For each job and machine size, all metaheuristics follow a similar pattern. Still, the differences between the confidence intervals of QL algorithms and traditional IG, IGALL increases as the job size increases. For 20 machines, it is obvious that the confidence intervals of the IGQ1, QIG, and IGQ2 algorithms do not intersect with the IG and IGALL algorithms' confidence intervals. Thus, QL algorithms statistically outperform the traditional IG and IGALL algorithms. Even the results of the IGALL algorithm are statistically significant to the traditional IG algorithm. For 40 and 60 machines, the results of the IGQ1, QIG, and IGQ2 algorithms are only meaningful when compared to the IG algorithm.

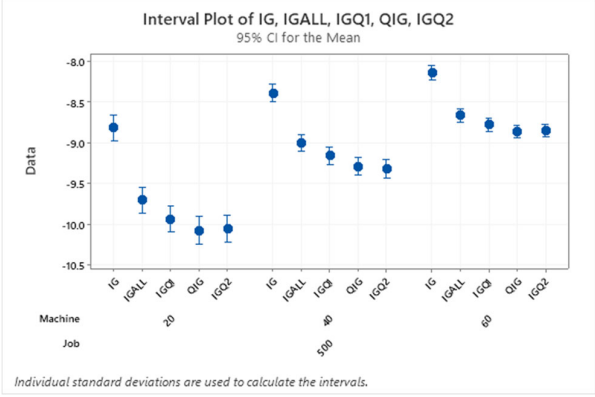

Fig. 6. Interval plot for 500 jobs and 20, 40, and 60 machines

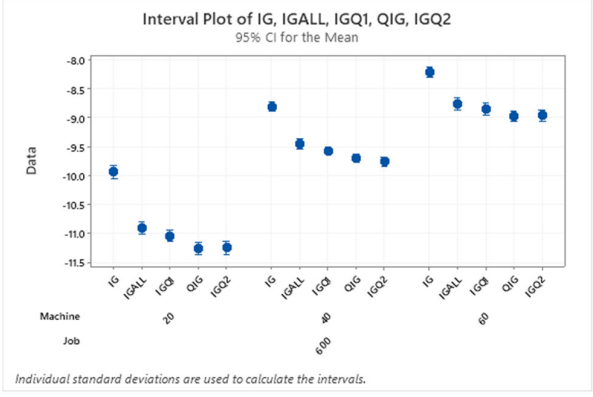

Fig. 7. Interval plot for 600 jobs and 20, 40, and 60 machines

Fig. 6 and 7 present the interval plot of metaheuristic algorithms for the 500 and 600 jobs, respectively, with 20, 40, and 60 machines under the 95% confidence interval. These two figures follow the same pattern. The traditional IG algorithms perform statistically worse than the other algorithms for all machine sizes. Also, the QIG and IGQ2 algorithms' confidence intervals do not intersect with the IGALL algorithm's, in all machine combinations, providing that their results are statistically better than the IGALL algorithm. Since the IGQ1 and IGALL algorithms have small intersections in all combinations, we cannot comment that their solutions are statistically different from each other. Fig. 8 and 9 present the interval plot of metaheuristic algorithms for the 700 and 800 jobs, respectively, with 20, 40, and 60 machines under the 95% confidence interval. These two figures also follow a similar pattern to the previous figures. Different than the previous figures, the QIG algorithm's confidence interval does not intersect with the other algorithms' except for the IGQ2 algorithm. These results

indicate that the best-performing algorithms are the QIG and the IGQ2 for the 700 and 800 jobs. The most significant difference generated by the traditional IG algorithm indicates that it is the worst-performing among all algorithms.

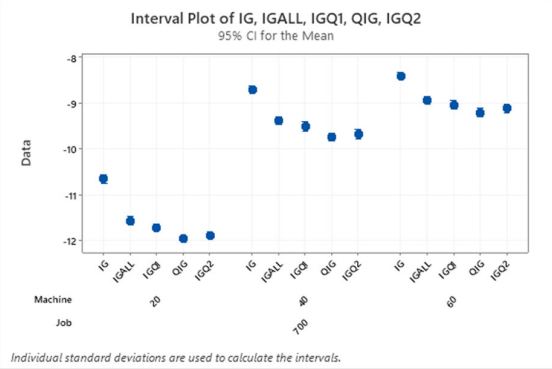

Fig. 8. Interval plot for 700 jobs and 20, 40, and 60 machines

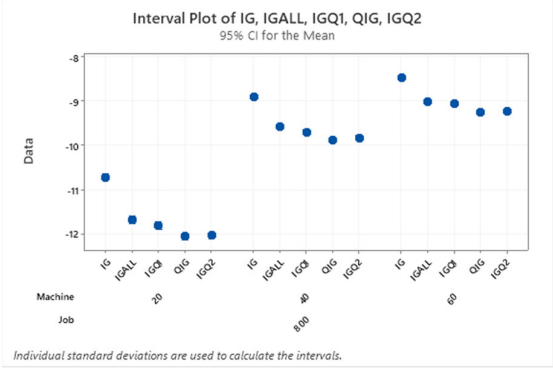

Fig. 9. Interval plot for 800 jobs and 20, 40, and 60 machines

From the above figures, when IG is compared to the IGALL algorithm, IGALL statistically performs better. Generally, we can claim that Q-learning-based algorithms generate the best results with respect to the traditional IG and IGALL algorithms. However, the results of the IGQ1, QIG, and IGQ2 algorithms are statistically not different than each other in many job and machine combinations. Last of all, it can be concluded that the proposed IG algorithms with Q-learning outperform the traditional IG algorithms and generate the best results.

7. Conclusion and future research

This study considers the BFSP to minimize makespan. Two types of mathematical models, such as MILP and CP, were developed to solve the problem and verify the results of the metaheuristic algorithms over the optimal solutions. Sets containing parameter values frequently used in the literature were created so that the parameter values of IG algorithms can be learned on their own while the algorithm is running. Then, by using the Q learning algorithm, a mechanism was developed to learn the parameter meter that is most suitable for the problem among the values in this set. Thus, besides the traditional IG and IGALL algorithms, IGQ1, QIG, and IGQ2 algorithms were developed. The performances of all models and metaheuristics were analyzed and compared using small and large-size VRF instances, and the best-known solutions were reported. In the analysis of the mathematical models, when the job size is 10, both models can find all the solutions optimally, but the CPU time of the MILP model is reasonably less than the CP model. However, as the job size increases from 20 to 60, both CP and MILP models have difficulty achieving optimal solutions within a 1-hour time limit. The CP model starts to perform better than the MILP model considering solution quality for larger job sizes. Thus, for the large-size VRF instances, only the results of the CP model were obtained to get comparisons with the metaheuristics. When the metaheuristic algorithms are compared over small VRF instances, all the algorithms perform similarly except the traditional IG algorithm, which is the statistically worst-performing algorithm of the other algorithms. Similar results were obtained for the large-size VRF instances. These results indicate that Q-learning-based IG algorithms are not statistically different than each other, but they perform better than the traditional IG and IGALL algorithms.

This study proves the robust performance of Q-learning-based IG algorithms on BFSP. In future studies, these algorithms can contribute to obtaining better results by applying them to different scheduling problems. In addition, Q-learning-based different metaheuristic algorithms can be developed, such as Q-learning-based iterated local search, variable neighborhood search, and so on. Self-adaptive learning of the parameter values of the algorithms will perform successfully on scheduling problems. Apart from the makespan objective function, its effect should be investigated by testing it on varied objective functions, i.e., total flow time or tardiness minimization. There are gaps in literature in this area, and we believe the literature will move in this direction in the future.

References

Aqil, S., & Allali, K. (2021). Two efficient nature inspired meta-heuristics solving blocking hybrid flow shop manufacturing problem. *Engineering Applications of Artificial Intelligence*, *100*, 104196. https://doi.org/10.1016/j.engappai.2021.104196

Birgin, E. G., Ferreira, J. E., & Ronconi, D. P. (2020). A filtered beam search method for the m-machine permutation flowshop scheduling problem minimizing the earliness and tardiness penalties and the waiting time of the jobs. *Computers and Operations Research*, *114*, 104824. https://doi.org/10.1016/j.cor.2019.104824

Blazewicz, J., H. Ecker, K., Pesch, E., Schmidt, G., & Węglarz, J. (2007). Handbook on scheduling. From theory to applications. *International Handbook on Information Systems*. https://doi.org/10.1007/978-3-540-32220-7

Caraffa, V., Ianes, S., P. Bagchi, T., & Sriskandarajah, C. (2001). Minimizing makespan in a blocking flowshop using

genetic algorithms. *International Journal of Production Economics*, *70*(2), 101–115. https://doi.org/10.1016/S0925-5273(99)00104-8

Carlier, J., Haouari, M., Kharbeche, M., & Moukrim, A. (2010). An optimization-based heuristic for the robotic cell problem. *European Journal of Operational Research*, *202*(3), 636–645. https://doi.org/10.1016/j.ejor.2009.06.035

Chen, H., Zhou, S., Li, X., & Xu, R. (2014). A hybrid differential evolution algorithm for a two-stage flow shop on batch processing machines with arbitrary release times and blocking. *International Journal of Production Research*, *52*(19), 5714–5734. https://doi.org/10.1080/00207543.2014.910625

Cheng, C.-Y., Ying, K.-C., Chen, H.-H., & Lu, H.-S. (2019). Minimising makespan in distributed mixed no-idle flowshops. *International Journal of Production Research*, *57*(1), 48–60. https://doi.org/10.1080/00207543.2018.1457812

Dubois-Lacoste, J., Pagnozzi, F., & Stützle, T. (2017). An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. *Computers & Operations Research*, *81*, 160–166. https://doi.org/10.1016/J.COR.2016.12.021

Elmi, A., & Topaloglu, S. (2013). A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. *Computers and Operations Research*, *40*(10), 2543–2555. https://doi.org/10.1016/j.cor.2013.01.024

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

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Math. Oper. Res.*, *1*(2), 117–129. https://doi.org/10.1287/moor.1.2.117

Gilmore, P. C., Lawler, E. L., & Shmoys, D. B. (1984). *Well-solved Special Cases of the Traveling Salesman Problem*. University of California at Berkeley.

Glover, F. (1990). Tabu search—part II. *ORSA Journal on Computing*, *2*, 4–32. https://doi.org/10.1287/ijoc.2.1.4

Gong, D., Han, Y., & Sun, J. (2018). A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems. *Knowledge-Based Systems*, *148*, 115–130. https://doi.org/10.1016/j.knosys.2018.02.029

Gong, H., Tang, L., & Duin, C. W. (2010). A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. *Computers and Operations Research*, *37*(5), 960–969. https://doi.org/10.1016/j.cor.2009.08.001

Hall, N. G., & Sriskandarajah, C. (1996). A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process. *Oper. Res.*, *44*(3), 510–525. https://doi.org/10.1287/opre.44.3.510

Han, Y.-Y., Gong, D., & Sun, X. (2015). A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. *Engineering Optimization*, *47*(7), 927–946. https://doi.org/10.1080/0305215X.2014.928817

Han, Y.-Y., Liang, J. J., Pan, Q.-K., Li, J.-Q., Sang, H.-Y., & Cao, N. N. (2013). Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem. *The International Journal of Advanced Manufacturing Technology*, *67*(1), 397–414. https://doi.org/10.1007/s00170-012-4493-5

Han, Y.-Y., Pan, Q.-K., Li, J.-Q., & Sang, H. (2012). An improved artificial bee colony algorithm for the blocking flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *60*(9), 1149–1159. https://doi.org/10.1007/s00170-011-3680-0

Han, Y., Gong, D., Li, J., & Zhang, Y. (2016). Solving the blocking flow shop scheduling problem with makespan using a modified fruit fly optimisation algorithm. *International Journal of Production Research*, *54*(22), 6782–6797. https://doi.org/10.1080/00207543.2016.1177671

Han, Y., Li, J., Sang, H., Liu, Y., Gao, K., & Pan, Q. (2020). Discrete evolutionary multi-objective optimization for energy-efficient blocking flow shop scheduling with setup time. *Applied Soft Computing Journal*, *93*, 106343. https://doi.org/10.1016/j.asoc.2020.106343

Johnson, S. M. (1954). Optimal Two and Three Stage Production Schedules With Set-Up Time Included. *Naval Research Logistics Quarterly*, *1*, 61–68. https://doi.org/10.1002/nav.3800010110

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *J. Artif. Int. Res.*, *4*(1), 237–285.

Karimi-Mamaghan, M., Mohammadi, M., Pasdeloup, B., & Meyer, P. (2022). 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*. https://doi.org/10.1016/j.ejor.2022.03.054

Kizilay, D., Tasgetiren, M. F., Pan, Q. K., & Gao, L. (2019). A variable block insertion heuristic for solving permutation flow shop scheduling problem with makespan criterion. *Algorithms*, *12*(5). https://doi.org/10.3390/a12050100

Mccormick, S., Pinedo, M., J. Shenker, S., & Wolf, B. (1989). Sequencing in an Assembly Line With Blocking to Minimize Cycle Time. *Operations Research*, *37*, 925–935. https://doi.org/10.1287/opre.37.6.925

Merchan, A. F., & Maravelias, C. T. (2016). Preprocessing and tightening methods for time-indexed MIP chemical production scheduling models. *Computers and Chemical Engineering*, *84*, 516–535. https://doi.org/10.1016/j.compchemeng.2015.10.003

Miyata, H. H., & Nagano, M. S. (2019). The blocking flow shop scheduling problem: A comprehensive and conceptual review. In *Expert Systems with Applications* (Vol. 137, pp. 130–156). Elsevier Ltd. https://doi.org/10.1016/j.eswa.2019.06.069

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, *37*(4), 754–768. https://doi.org/10.1016/J.COR.2009.06.019

Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91–95. https://doi.org/10.1016/0305-0483(83)90088-9

Newton, M. A. H., Riahi, V., Su, K., & Sattar, A. (2019). Scheduling blocking flowshops with setup times via constraint guided and accelerated local search. *Computers & Operations Research*, *109*, 64–76. https://doi.org/10.1016/J.COR.2019.04.024

Osman, I., & N. Potts, C. (1989). Simulated Annealing for Permutation Flow-Shop Scheduling. *Omega*, *17*, 551–557. https://doi.org/10.1016/0305-0483(89)90059-5

Öztop, H, Tasgetiren, M. F., Kandiller, L., & Pan, Q.-K. (2020). A Novel General Variable Neighborhood Search through Q-Learning for No-Idle Flowshop Scheduling. *2020 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. https://doi.org/10.1109/CEC48606.2020.9185556

Öztop, Hande, Tasgetiren, M. F., Kandiller, L., & Pan, Q.-K. (2022). Metaheuristics with restart and learning mechanisms for the no-idle flowshop scheduling problem with makespan criterion. *Computers & Operations Research*, *138*, 105616. https://doi.org/https://doi.org/10.1016/j.cor.2021.105616

Pan, Q.-K., & Ruiz, R. (2012). An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. *Omega*, *40*(2), 166–180. https://doi.org/10.1016/J.OMEGA.2011.05.002

Ramezanian, R., Vali-Siar, M. M., & Jalalian, M. (2019). Green permutation flowshop scheduling problem with sequence-dependent setup times: a case study. *International Journal of Production Research*, *57*(10), 3311–3333. https://doi.org/10.1080/00207543.2019.1581955

Riahi, V., Newton, M. A. H., Su, K., & Sattar, A. (2019). Constraint guided accelerated search for mixed blocking permutation flowshop scheduling. *Computers & Operations Research*, *102*, 102–120. https://doi.org/10.1016/J.COR.2018.10.003

Ribas, I., & Companys, R. (2015). Efficient heuristic algorithms for the blocking flow shop scheduling problem with total flow time minimization. *Computers & Industrial Engineering*, *87*, 30–39. https://doi.org/https://doi.org/10.1016/j.cie.2015.04.013

Ribas, I., Companys, R., & Tort-Martorell, X. (2015). An efficient Discrete Artificial Bee Colony algorithm for the blocking flow shop problem with total flowtime minimization. *Expert Systems with Applications*, *42*(15), 6155–6167. https://doi.org/https://doi.org/10.1016/j.eswa.2015.03.026

Ribas, I., Companys, R., & Tort-Martorell, X. (2017). Efficient heuristics for the parallel blocking flow shop scheduling problem. *Expert Systems with Applications*, *74*, 41–54. https://doi.org/10.1016/j.eswa.2017.01.006

Ribas, I., Companys, R., & Tort-Martorell, X. (2019). An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem. *Expert Systems with Applications*, *121*, 347–361. https://doi.org/10.1016/j.eswa.2018.12.039

Ronconi, D P, & Armentano, V. A. (2001). Lower bounding schemes for flowshops with blocking in-process. *Journal of the Operational Research Society*, *52*(11), 1289–1297. https://doi.org/10.1057/palgrave.jors.2601220

Ronconi, Débora P. (2004). A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, *87*(1), 39–48. https://doi.org/10.1016/S0925-5273(03)00065-3

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/https://doi.org/10.1016/j.ejor.2005.12.009

Shao, Z., Pi, D., & Shao, W. (2018a). Estimation of distribution algorithm with path relinking for the blocking flow-shop scheduling problem. *Engineering Optimization*, *50*(5), 894–916. https://doi.org/10.1080/0305215X.2017.1353090

Shao, Z., Pi, D., & Shao, W. (2018b). A novel discrete water wave optimization algorithm for blocking flow-shop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation*, *40*, 53–75. https://doi.org/10.1016/j.swevo.2017.12.005

Shao, Z., Pi, D., & Shao, W. (2020). Hybrid enhanced discrete fruit fly optimization algorithm for scheduling blocking flow-shop in distributed environment. *Expert Systems with Applications*, *145*, 113147. https://doi.org/https://doi.org/10.1016/j.eswa.2019.113147

Shao, Z., Pi, D., Shao, W., & Yuan, P. (2019). An efficient discrete invasive weed optimization for blocking flow-shop scheduling problem. *Engineering Applications of Artificial Intelligence*, *78*, 124–141. https://doi.org/10.1016/j.engappai.2018.11.005

Shao, Z., Shao, W., & Pi, D. (2020). Effective heuristics and metaheuristics for the distributed fuzzy blocking flow-shop scheduling problem. *Swarm and Evolutionary Computation*, *59*, 100747. https://doi.org/10.1016/j.swevo.2020.100747

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (Second Edi). The MIT Press.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, *47*(1), 65–74. https://doi.org/10.1016/0377-2217(90)90090-X

Tasgetiren, M. Fatih, Kizilay, D., Pan, Q. K., & Suganthan, P. N. (2017). Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. *Computers and Operations Research*, *77*. https://doi.org/10.1016/j.cor.2016.07.002

Tasgetiren, M. Fatih, Pan, Q.-K., & Liang, Y.-C. (2009). A discrete differential evolution algorithm for the single machine

total weighted tardiness problem with sequence dependent setup times. *Computers & Operations Research*, *36*(6), 1900–1915. https://doi.org/https://doi.org/10.1016/j.cor.2008.06.007

Tasgetiren, M.F., Pan, Q.-K., Kizilay, D., & Gao, K. (2016). A Variable Block Insertion Heuristic for the Blocking Flowshop Scheduling Problem with Total Flowtime Criterion. *Algorithms*, *9*(4). https://doi.org/10.3390/a9040071

Trabelsi, W., Sauvey, C., & Sauer, N. (2012). Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems. *Computers & Operations Research*, *39*(11), 2520–2527. https://doi.org/https://doi.org/10.1016/j.cor.2011.12.022

Vallada, E., Ruiz, R., & Framinan, J. M. (2015). New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, *240*(3), 666–677. https://doi.org/https://doi.org/10.1016/j.ejor.2014.07.033

Wang, L., Pan, Q.-K., & Fatih Tasgetiren, M. (2010). Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. *Expert Systems with Applications*, *37*(12), 7929–7936. https://doi.org/10.1016/J.ESWA.2010.04.042

Wang, L., Pan, Q.-K., Suganthan, P. N., Wang, W.-H., & Wang, Y.-M. (2010). A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research*, *37*(3), 509–520. https://doi.org/10.1016/J.COR.2008.12.004

Wang, X., & Tang, L. (2012). A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking. *Applied Soft Computing Journal*, *12*(2), 652–662. https://doi.org/10.1016/j.asoc.2011.09.021

Watkins, C. (1989). *Learning From Delayed Rewards* [King's College]. http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

Yuan, S., Li, T., & Wang, B. (2020). A co-evolutionary genetic algorithm for the two-machine flow shop group scheduling problem with job-related blocking and transportation times. *Expert Systems with Applications*, *152*, 113360. https://doi.org/10.1016/j.eswa.2020.113360

Zhang, G., Xing, K., & Cao, F. (2018). Discrete differential evolution algorithm for distributed blocking flowshop scheduling with makespan criterion. *Engineering Applications of Artificial Intelligence*, *76*, 96–107. https://doi.org/10.1016/J.ENGAPPAI.2018.09.005

Appendix

A1. Best Results for the BFSP on Small VRF Benchmark

Table A1
Best BFSP Results for Small VRF Benchmark

Ins.	BKS	Ins.	BKS	Ins.	BKS	Ins.	BKS	Ins.	BKS	Ins.	BKS
10_5_1	**716**	20_5_1	**1300**	30_5_1	**1976**	40_5_1	**2630**	50_5_1	**3344**	60_5_1	**3668**
10_5_2	**739**	20_5_2	**1377**	30_5_2	**1780**	40_5_2	**2649**	50_5_2	**3044**	60_5_2	**3501**
10_5_3	**770**	20_5_3	**1451**	30_5_3	**1894**	40_5_3	**2413**	50_5_3	**3104**	60_5_3	**3652**
10_5_4	**742**	20_5_4	**1270**	30_5_4	**1947**	40_5_4	**2430**	50_5_4	**3159**	60_5_4	**3674**
10_5_5	**783**	20_5_5	**1434**	30_5_5	**1844**	40_5_5	**2567**	50_5_5	**3144**	60_5_5	**3620**
10_5_6	**779**	20_5_6	**1237**	30_5_6	**1960**	40_5_6	**2403**	50_5_6	**3058**	60_5_6	**3526**
10_5_7	**784**	20_5_7	**1288**	30_5_7	**1856**	40_5_7	**2496**	50_5_7	**2859**	60_5_7	**3701**
10_5_8	**722**	20_5_8	**1235**	30_5_8	**1962**	40_5_8	**2622**	50_5_8	**2989**	60_5_8	**3830**
10_5_9	**798**	20_5_9	**1389**	30_5_9	**1926**	40_5_9	**2563**	50_5_9	**2946**	60_5_9	**3547**
10_5_10	**691**	20_5_10	**1415**	30_5_10	**1890**	40_5_10	**2577**	50_5_10	**3127**	60_5_10	**3776**
10_10_1	**1173**	20_10_1	**1699**	30_10_1	**2233**	40_10_1	**2934**	50_10_1	**3525**	60_10_1	**4202**
10_10_2	**1178**	20_10_2	**1692**	30_10_2	**2384**	40_10_2	**2886**	50_10_2	**3502**	60_10_2	**4295**
10_10_3	**1146**	20_10_3	**1769**	30_10_3	**2367**	40_10_3	**2869**	50_10_3	**3570**	60_10_3	**4143**
10_10_4	**1081**	20_10_4	**1620**	30_10_4	**2275**	40_10_4	**2986**	50_10_4	**3615**	60_10_4	**4200**
10_10_5	**1132**	20_10_5	**1767**	30_10_5	**2337**	40_10_5	**2919**	50_10_5	**3713**	60_10_5	**4318**
10_10_6	**1108**	20_10_6	**1765**	30_10_6	**2362**	40_10_6	**2974**	50_10_6	**3536**	60_10_6	**4314**
10_10_7	**1164**	20_10_7	**1734**	30_10_7	**2283**	40_10_7	**2928**	50_10_7	**3455**	60_10_7	**4353**
10_10_8	**1149**	20_10_8	**1724**	30_10_8	**2181**	40_10_8	**2901**	50_10_8	**3690**	60_10_8	**4163**
10_10_9	**1088**	20_10_9	**1697**	30_10_9	**2197**	40_10_9	**2871**	50_10_9	**3548**	60_10_9	**4231**
10_10_10	**1180**	20_10_10	**1679**	30_10_10	**2256**	40_10_10	**2933**	50_10_10	**3573**	60_10_10	**4295**
10_15_1	**1337**	20_15_1	**2078**	30_15_1	**2669**	40_15_1	**3427**	50_15_1	**3909**	60_15_1	**4680**
10_15_2	**1421**	20_15_2	**2071**	30_15_2	**2596**	40_15_2	**3285**	50_15_2	**3945**	60_15_2	**4647**
10_15_3	**1481**	20_15_3	**1957**	30_15_3	**2582**	40_15_3	**3347**	50_15_3	**3983**	60_15_3	**4682**
10_15_4	**1508**	20_15_4	**1999**	30_15_4	**2703**	40_15_4	**3329**	50_15_4	**4134**	60_15_4	**4480**
10_15_5	**1425**	20_15_5	**2007**	30_15_5	**2739**	40_15_5	**3372**	50_15_5	**3983**	60_15_5	**4640**
10_15_6	**1379**	20_15_6	**2145**	30_15_6	**2628**	40_15_6	**3202**	50_15_6	**4001**	60_15_6	**4776**
10_15_7	**1461**	20_15_7	**2176**	30_15_7	**2604**	40_15_7	**3330**	50_15_7	**4157**	60_15_7	**4612**
10_15_8	**1514**	20_15_8	**1982**	30_15_8	**2694**	40_15_8	**3402**	50_15_8	**4050**	60_15_8	**4586**
10_15_9	**1533**	20_15_9	**2057**	30_15_9	**2492**	40_15_9	**3203**	50_15_9	**3866**	60_15_9	**4542**
10_15_10	**1496**	20_15_10	**2037**	30_15_10	**2694**	40_15_10	**3349**	50_15_10	**4049**	60_15_10	**4717**
10_20_1	**1697**	20_20_1	**2463**	30_20_1	**2941**	40_20_1	**3691**	50_20_1	**4331**	60_20_1	**4998**
10_20_2	**1800**	20_20_2	**2339**	30_20_2	**3139**	40_20_2	**3680**	50_20_2	**4317**	60_20_2	**5020**
10_20_3	**1779**	20_20_3	**2416**	30_20_3	**3088**	40_20_3	**3658**	50_20_3	**4421**	60_20_3	**5131**
10_20_4	**1719**	20_20_4	**2305**	30_20_4	**2970**	40_20_4	**3684**	50_20_4	**4354**	60_20_4	**5002**
10_20_5	**1733**	20_20_5	**2360**	30_20_5	**2981**	40_20_5	**3520**	50_20_5	**4248**	60_20_5	**4944**
10_20_6	**1927**	20_20_6	**2445**	30_20_6	**3019**	40_20_6	**3603**	50_20_6	**4387**	60_20_6	**4979**
10_20_7	**1728**	20_20_7	**2449**	30_20_7	**3034**	40_20_7	**3673**	50_20_7	**4297**	60_20_7	**5038**
10_20_8	**1707**	20_20_8	**2322**	30_20_8	**3079**	40_20_8	**3664**	50_20_8	**4376**	60_20_8	**4978**
10_20_9	**1740**	20_20_9	**2525**	30_20_9	**3090**	40_20_9	**3748**	50_20_9	**4418**	60_20_9	**4975**
10_20_10	**1693**	20_20_10	**2337**	30_20_10	**3108**	40_20_10	**3553**	50_20_10	**4389**	60_20_10	**4936**

A2. Best Results for the BFSP on Large VRF Benchmark

Table A2
Best BFSP Results for Large VRF Benchmark

Ins.	BKS	Ins.	BKS	Ins.	BKS	Ins.	BKS	Ins.	BKS	Ins.	BKS	Ins.	BKS	Ins.	BKS
100_20_1	**7755**	200_20_1	**14782**	300_20_1	**21748**	400_20_1	**28755**	500_20_1	**36114**	600_20_1	**43228**	700_20_1	**50239**	800_20_1	**57563**
100_20_2	**7832**	200_20_2	**14784**	300_20_2	**21894**	400_20_2	**29152**	500_20_2	**36404**	600_20_2	**43070**	700_20_2	**50178**	800_20_2	**57412**
100_20_3	**7733**	200_20_3	**14923**	300_20_3	**21716**	400_20_3	**29050**	500_20_3	**36147**	600_20_3	**43368**	700_20_3	**50138**	800_20_3	**57423**
100_20_4	**7774**	200_20_4	**14699**	300_20_4	**21757**	400_20_4	**28808**	500_20_4	**35817**	600_20_4	**43104**	700_20_4	**50136**	800_20_4	**57437**
100_20_5	**7803**	200_20_5	**14651**	300_20_5	**21911**	400_20_5	**28955**	500_20_5	**36239**	600_20_5	**43106**	700_20_5	**50475**	800_20_5	**57498**
100_20_6	**7831**	200_20_6	**14914**	300_20_6	**21770**	400_20_6	**29006**	500_20_6	**35905**	600_20_6	**43319**	700_20_6	**50434**	800_20_6	**57790**
100_20_7	**7948**	200_20_7	**14797**	300_20_7	**21700**	400_20_7	**29094**	500_20_7	**36092**	600_20_7	**42905**	700_20_7	**50504**	800_20_7	**57130**
100_20_8	**7681**	200_20_8	**14671**	300_20_8	**21925**	400_20_8	**28866**	500_20_8	**36167**	600_20_8	**43261**	700_20_8	**50322**	800_20_8	**57282**
100_20_9	**7837**	200_20_9	**14640**	300_20_9	**21772**	400_20_9	**28764**	500_20_9	**36120**	600_20_9	**43317**	700_20_9	**50162**	800_20_9	**57256**
100_20_10	**7684**	200_20_10	**14824**	300_20_10	**22114**	400_20_10	**28980**	500_20_10	**35768**	600_20_10	**42861**	700_20_10	**49982**	800_20_10	**57374**
100_40_1	**9193**	200_40_1	**16614**	300_40_1	**24111**	400_40_1	**31632**	500_40_1	**39112**	600_40_1	**46790**	700_40_1	**54437**	800_40_1	**61964**
100_40_2	**9395**	200_40_2	**16637**	300_40_2	**24244**	400_40_2	**31630**	500_40_2	**39348**	600_40_2	**46650**	700_40_2	**54371**	800_40_2	**61761**
100_40_3	**9300**	200_40_3	**16746**	300_40_3	**24226**	400_40_3	**31779**	500_40_3	**39216**	600_40_3	**46903**	700_40_3	**54320**	800_40_3	**61972**
100_40_4	**9221**	200_40_4	**16708**	300_40_4	**24071**	400_40_4	**31724**	500_40_4	**39261**	600_40_4	**46806**	700_40_4	**54571**	800_40_4	**61891**
100_40_5	**9387**	200_40_5	**16619**	300_40_5	**24270**	400_40_5	**31550**	500_40_5	**39355**	600_40_5	**46715**	700_40_5	**54278**	800_40_5	**62030**
100_40_6	**9351**	200_40_6	**16645**	300_40_6	**24210**	400_40_6	**31699**	500_40_6	**39121**	600_40_6	**46847**	700_40_6	**54228**	800_40_6	**61832**
100_40_7	**9342**	200_40_7	**16690**	300_40_7	**24152**	400_40_7	**31727**	500_40_7	**39277**	600_40_7	**46895**	700_40_7	**54265**	800_40_7	**61667**
100_40_8	**9321**	200_40_8	**16629**	300_40_8	**24178**	400_40_8	**31637**	500_40_8	**39315**	600_40_8	**46606**	700_40_8	**54564**	800_40_8	**62000**
100_40_9	**9279**	200_40_9	**16562**	300_40_9	**24079**	400_40_9	**31833**	500_40_9	**39516**	600_40_9	**46886**	700_40_9	**54425**	800_40_9	**62008**
100_40_10	**9256**	200_40_10	**16726**	300_40_10	**24119**	400_40_10	**31745**	500_40_10	**39434**	600_40_10	**46844**	700_40_10	**54361**	800_40_10	**61903**
100_60_1	**1055**	200_60_1	**18282**	300_60_1	**25895**	400_60_1	**33549**	500_60_1	**41320**	600_60_1	**49047**	700_60_1	**56906**	800_60_1	**64568**
100_60_2	**10795**	200_60_2	**18228**	300_60_2	**25831**	400_60_2	**33578**	500_60_2	**41111**	600_60_2	**49017**	700_60_2	**56655**	800_60_2	**64455**
100_60_3	**10491**	200_60_3	**18369**	300_60_3	**25877**	400_60_3	**33702**	500_60_3	**41326**	600_60_3	**49043**	700_60_3	**56654**	800_60_3	**64491**
100_60_4	**10574**	200_60_4	**18078**	300_60_4	**25874**	400_60_4	**33683**	500_60_4	**41378**	600_60_4	**49025**	700_60_4	**56786**	800_60_4	**64413**
100_60_5	**10607**	200_60_5	**18186**	300_60_5	**25781**	400_60_5	**33539**	500_60_5	**41334**	600_60_5	**49087**	700_60_5	**56810**	800_60_5	**64374**
100_60_6	**10843**	200_60_6	**18286**	300_60_6	**25770**	400_60_6	**33462**	500_60_6	**41311**	600_60_6	**48992**	700_60_6	**56714**	800_60_6	**64704**
100_60_7	**10554**	200_60_7	**18264**	300_60_7	**26019**	400_60_7	**33537**	500_60_7	**41266**	600_60_7	**48962**	700_60_7	**56362**	800_60_7	**64438**
100_60_8	**10754**	200_60_8	**18200**	300_60_8	**25866**	400_60_8	**33565**	500_60_8	**41445**	600_60_8	**49091**	700_60_8	**56740**	800_60_8	**64505**
100_60_9	**10687**	200_60_9	**18139**	300_60_9	**25991**	400_60_9	**33768**	500_60_9	**41246**	600_60_9	**49168**	700_60_9	**56596**	800_60_9	**64607**
100_60_10	**10715**	200_60_10	**18121**	300_60_10	**25988**	400_60_10	**33608**	500_60_10	**41360**	600_60_10	**49010**	700_60_10	**57033**	800_60_10	**64320**