Applied Soft Computing Journal 97 (2020) 106790

Contents lists available at ScienceDirect

Applied Soft Computing Journal

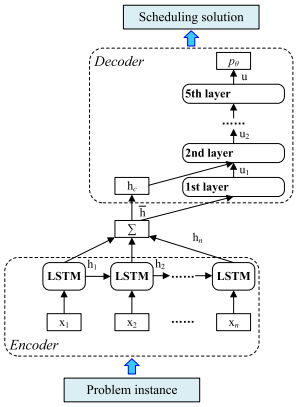
journal homepage: www.elsevier.com/locate/asoc

Real-time neural network scheduling of emergency medical mask production during COVID-19

Chen-Xin Wu [a], Min-Hui Liao [a], Mumtaz Karatas [b], Sheng-Yong Chen [c], Yu-Jun Zheng [a,*]

[a] School of Information Science and Engineering, Hangzhou Normal University, Hangzhou 311121, China
[b] Industrial Engineering Department, Naval Academy, National Defense University, Tuzla 34940, Istanbul, Turkey
[c] School of Computer Science and Engineering, Tianjin University of Technology, Tianjin 300384, China

GRAPHICAL ABSTRACT



ARTICLE INFO

Article history:
Received 16 August 2020
Received in revised form 17 September 2020
Accepted 6 October 2020
Available online 14 October 2020

Keywords:
Emergency production
Flow shop scheduling
Neural network
Reinforcement learning
Public health emergencies

ABSTRACT

During the outbreak of the novel coronavirus pneumonia (COVID-19), there is a huge demand for medical masks. A mask manufacturer often receives a large amount of orders that must be processed within a short response time. It is of critical importance for the manufacturer to schedule and reschedule mask production tasks as efficiently as possible. However, when the number of tasks is large, most existing scheduling algorithms require very long computational time and, therefore, cannot meet the needs of emergency response. In this paper, we propose an end-to-end neural network, which takes a sequence of production tasks as inputs and produces a schedule of tasks in a real-time manner. The network is trained by reinforcement learning using the negative total tardiness as the reward signal. We applied the proposed approach to schedule emergency production tasks for a medical mask manufacturer during the peak of COVID-19 in China. Computational results show that the neural network scheduler can solve problem instances with hundreds of tasks within seconds. The objective function value obtained by the neural network scheduler is significantly better than those of existing constructive heuristics, and is close to those of the state-of-the-art metaheuristics whose computational time is unaffordable in practice.

1. Introduction

Since the outbreak of the novel coronavirus pneumonia (COVID-19), there has been an ever-growing demand for medical masks. A mask manufacturer often receives a large amount of

* Corresponding author.
E-mail address: yujun.zheng@computer.org (Y.-J. Zheng).

https://doi.org/10.1016/j.asoc.2020.106790
1568-4946/

orders that must be processed within a short response time. Therefore, it is of critical importance for the manufacturer to schedule mask production tasks as efficiently as possible. Moreover, as new orders arrive continuously, the manufacturer also need to reschedule production tasks frequently. This brings a great challenge for the manufacturer to produce high-quality scheduling solutions rapidly.

The motivation of this paper comes from our cooperation with ZHENDE company, a medical apparatus manufacturer in Zhejiang Province, China, during COVID-19. The manufacturer has a mask production line that can produce different types masks, such as disposable medical masks, surgical masks, medical protective masks, and respiratory masks. The daily output is nearly one hundred thousand. However, on each day since the outbreak of COVID-19, it receives tens to hundreds of orders, the total demand of which ranges from hundreds of thousands to a million masks, and almost all orders have tight delivery deadlines. The manufacturer asked our research team to develop a production scheduler that can schedule hundreds of tasks within seconds. During the COVID-19 pandemic, a lot of medical supply manufacturers having similar requirements of production task scheduling.

Scheduling production tasks on a production line can be formulated as a machine scheduling problem which is known to be NP-hard [1]. Exact optimization algorithms (e.g., [2–5]) often have very large computational times that are infeasible on even moderate-size problem instances. As for moderate- and large-size instances optimal solutions are rarely needed in practice, heuristic approximation algorithms, including constructive heuristics (e.g., [6–8]) and metaheuristic evolutionary algorithms (e.g., [9–15]), are more feasible to achieve a trade-off between optimality and computational costs. However, the solution fitness of constructive heuristics is often low for even moderate-size instances. For metaheuristics, the number of repeated generations and objective function evaluations for solving large-size instances still takes a relatively long time and, therefore, cannot satisfy the requirement of real-time scheduling. Table 1 gives an overview of main classes of algorithms for NP-hard production scheduling problems.

In this paper, we propose a deep reinforcement approach for scheduling real-time production tasks. The scheduler is a deep neural network (DNN), which consists of an encoder that takes a sequence of production tasks as inputs to predict a distribution over different schedules, and a decoder that predict the probability of selecting a task into the schedule at each time step. The network parameters are optimized by reinforcement learning using the negative total tardiness as the reward signal. After being trained on sufficient (unlabeled) instances following the distribution of the problem, the network can directly produce a solution for a new instance within a very short time. We applied the DNN scheduler to mask production scheduling in the ZHENDE manufacturer during the peak of COVID-19 in China. Computational results show that the proposed method can solve problem instances with hundreds of tasks within seconds. The objective function value (total weighted tardiness) produced by the DNN scheduler is significantly better than those of existing constructive heuristics such as the Nawaz, Enscore and Ham (NEH) heuristic [6] and Suliman heuristic [7], and is very close to those of the state-of-the-art metaheuristics whose computational time is obviously unaffordable in practice.

The remainder of this paper is organized as follows. Section 2 briefly reviews the related work on machine learning methods for scheduling problems. Section 3 describes the considered emergency production scheduling problem. Section 4 describes the DNN scheduler in detail, including the model architecture and learning algorithm. Section 5 presents the experimental results, and finally Section Section 6 concludes with a discussion.

2. Related work

To solve NP-hard combinatorial optimization problems, classical algorithms, including exact optimization algorithms, heuristic optimization algorithms, and metaheuristic algorithms, are essentially search algorithms that explore the solution spaces to find optimal or near-optimal solutions [8]. Using end-to-end neural networks to directly map a problem input to a solution is another research direction that has received increasing attention [17]. The earliest work dates back to Hopfield and Tank [18], who applied a Hopfield-network to solve the traveling salesman problem (TSP). Simon and Takefuji [19] modified the Hopfield network to solve the job-shop scheduling problem. However, the Hopfield network is only suitable for very small problem instances. Based on the premise that optimal solutions to a scheduling problem have common features which can be implicitly captured by machine learning, Weckman et al. [20] proposed a neural network for scheduling job-shops by capturing the predictive knowledge regarding the assignment of operation's position in a sequence. They used solutions obtained by genetic algorithm (GA) as samples for training the network. To solve the flow shop scheduling problem, Ramanan et al. [21] used a neural network trained with optimal solutions of known instances to produce quality solutions for new instances, which are then given as the initial solutions to improve other heuristics such as GA. Such methods combining machine learning and metaheuristics are still time-consuming on large problems.

Recently, deep learning has been utilized to optimization algorithm design by learning algorithmic decisions based on the distribution of problem instances. Vinyals et al. [22] introduced the pointer network as a sequence-to-sequence model, which consists in an encoder to parse the input nodes, and a decoder to produce a probability distribution over these nodes based on a pointer (attention) mechanism over the encoded nodes. They applied the pointer network to solve TSP instances with up to 100 nodes. However, the pointer network is trained in a supervised manner, which heavily relies on sample instances with known optimal solutions which are expensive to obtain. Nazari et al. [23] addressed this difficulty by using reinforcement learning to calculate the rewards of output solutions and introducing an attention mechanism to address different parts of the input. They applied the model to solve the vehicle routing problem (VRP). Kool et al. [16] used a different decoder based on a context vector and improved the training algorithm based on a greedy rollout baseline. They applied the model to several combinatorial optimization problems including TSP and VRP. For online scheduling of vehicle services in large transportation networks, Yu et al. [24] also employed reinforcement learning to train a deep graph embedded pointer network, which employs an auxiliary critic neural network to estimate the expected output. Peng et al. [25] presented a dynamic attention model with dynamic encoder–decoder architecture to exploit hidden structure information at different construction steps, so as to construct better solutions. Solozabal et al. [26] extended the neural combinatorial optimization approach to constrained combinatorial optimization, where solution decisions are inferred based on both the reward signal generated from objective function values and penalty signals generated from constraint violations. However, to our knowledge, studies on deep learning approaches for efficiently solving emergency production scheduling problems are still few.

3. Medical mask production scheduling problem

In this section, we formulate the scheduling problem as follows (the variables are listed in Table 2). The manufacturer has K

Table 1
Advantages and disadvantages of main classes of algorithms for NP-hard production scheduling problems.

	Solution quantity	Computational time	Typical work
Exact optimization algorithms	Optimality guarantee	Long (exponential in problem size)	[2–5]
Constructive heuristics	Low for moderate and large problems	Short (typically polynomial in problem size)	[6–8]
Metaheuristics	Near-optimal or high-quantity	Moderate (polynomial in the number of generations times problem size)	[9–15]
Neural optimization	Typically between constructive heuristics and metaheuristics	Short (polynomial in problem size)	[16,17]

orders, denoted by $\mathbf{O} = \{O_1, O_2, \ldots, O_K\}$, to be processed. Each order O_k is associated with a set Φ_k of production tasks (jobs), and each task specifies the required type and number of masks. Each order O_k has an expected delivery time d_k and an importance weight w_k defined according to its value and urgency. In our practice, the manager gives a score between 1–10 for each order, and then all weights are normalized such that $(\sum_{k=1}^{K} w_k) = 1$.

Let $\mathbf{J} = \{J_1, J_2, \ldots, J_n\}$ be the set of all tasks. These tasks need to be scheduled on a production line with m machines, denoted by $\mathbf{M} = \{M_1, M_2, \ldots, M_m\}$. Each task J_j has exactly m operations, where the ith operation must be processed on machine M_i with a processing time t_{ij} ($1 \le i \le m$; $1 \le j \le n$). Each machine can process at most one task at a time, and each operation cannot be interrupted. The operations of mask production typically include cloth cutting, fabric lamination, belt welding, disinfection, and packaging.

The problem is to decide a processing sequence $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_n\}$ of the n tasks. Let $C(\pi_i, j)$ denote the completion time of task π_j on machine M_i. For the first machine M_1, the tasks can be sequentially processed immediately one by one:

$$C(\pi_1, 1) = t_{\pi_1,1} \tag{1}$$

$$C(\pi_j, 1) = C(\pi_{j-1}, 1) + t_{\pi_j,1}, \quad j = 2, \ldots, n \tag{2}$$

The first job π_1 can be processed on each subsequent machine M_i immediately after it is completed on the previous machine M_{i-1}:

$$C(\pi_1, i) = C(\pi_1, i-1) + t_{\pi_1,i}, \quad i = 2, \ldots, m \tag{3}$$

Each subsequent job π_j can be processed on machine M_i only when (1) the job π_j has been completed on the previous machine M_{i-1}; (2) the previous job π_{j-1} has been completed on machine M_i:

$$C(\pi_j, i) = \max\big(C(\pi_j, i-1), C(\pi_{j-1}, i)\big) + t_{\pi_j,i}, \quad i = 2, \ldots, m; j = 2, \ldots, n \tag{4}$$

Therefore, the completion time of each order O_k is the completion time of the last task of the order on machine M_m:

$$T(O_k) = \max_{\pi \in \Phi_k} C(\pi, m) \tag{5}$$

The objective of the problem is to minimize the total weighted tardiness of the orders:

$$\min f(\boldsymbol{\pi}) = \sum_{k=1}^{K} w_k \max(T(O_k) - d_k, 0) \tag{6}$$

If all tasks are available for processing at time zero, the above formulation can be regarded as a variant of the permutation flow shop scheduling problem which is known to be NP-hard [1]. When there are hundreds of tasks to be scheduled, the problem instances are computationally intractable for exact optimization algorithms, and search-based heuristics also typically take tens of minutes to hours to obtain a satisfying solution. Moreover, in a public health emergency such as the COVID-19 pandemic, new orders may continually arrive during the emergency production and, therefore, it needs to frequently reschedule production tasks to incorporate new tasks into the schedules. The allowable computational time for rescheduling is even shorter, typical only a few seconds. Hence, it is required to design real-time or near-real-time rescheduling methods for the problem.

4. A neural network scheduler for emergence production task scheduling

We propose a neural network scheduler to efficiently solve the above production task scheduling problem. The network takes a sequence of production tasks as inputs, and produce a schedule of the tasks by sequentially selecting a task into the schedule at each time step. We first use reinforcement learning to train the network on a large number of unlabeled instances (exact optimal solutions of which are not needed), and then use supervised learning to fine tune the network on some labeled instances (exact optimal solutions of which are known). Given a new instance as the input to the network, it is expected to produce a high-quality scheduling solution if the training instances well represent the distribution of the scheduling problem. Since the network is put into use, we periodically collect these real-world instances and solve them with state-of-the-art metaheuristics and, thus, construct new labeled instances to re-train the network. The basic flow to apply our machine learning approach is illustrated in Fig. 1.

4.1. Neural network model

The proposed neural network scheduler is based on the encoder–decoder architecture [27]. Fig. 2 illustrates the architecture of the network. The input to the network is a problem instance represented by a sequence of n tasks, each of which is described by a $(m+2)$-dimensional vector $\mathbf{x}_j = \{p_{j,1}, p_{j,2}, \ldots, p_{j,m}, d_k, w_k\}$ that consists the processing times on the m machines and the expected delivery time and weight importance of the corresponding order. To facilitate the processing of the neural network, all inputs are normalized into [0,1], e.g., each d_k is transformed to $(d_k - d_{\min})/(d_{\max} - d_{\min})$, where $d_{\min} = \min_{1 \le k \le K} d_k$ and $d_{\max} = \max_{1 \le k \le K} d_k$.

The encoder is a recurrent neural networks (RNN) with long short-term memory (LSTM) [28] cells. An LSTM takes a task $\mathbf{x}_j$ as input at a time and transforms it to a hidden state $\mathbf{h}_j$ by increasingly computing the embedding of the inputs (where *att* denotes the transformation by LSTM):

$$\begin{aligned} \mathbf{h}_1 &= att(\mathbf{h}_1, \mathbf{x}_1) = encode(\mathbf{x}_1) \\ \mathbf{h}_2 &= att(\mathbf{h}_1, \mathbf{x}_2) = encode(\mathbf{x}_1, \mathbf{x}_2) \\ &\vdots \\ \mathbf{h}_n &= att(\mathbf{h}_{n-1}, \mathbf{x}_n) = encode(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \end{aligned} \tag{7}$$

Table 2
Mathematical variables used in the problem formulation.

Symbol	Description
$\mathbf{O} = \{O_1, O_2, \ldots, O_K\}$	The set of orders
K	Number of orders
k	Index of orders ($1 \le k \le K$)
d_k	Expected delivery time of order O_k
w_k	Importance weight of order O_k
Φ_k	Set of production tasks in order O_k
$\mathbf{J} = \{J_1, J_2, \ldots, J_n\}$	Set of all production tasks
n	Number of tasks
j	Index of tasks ($1 \le j \le n$)
$\mathbf{M} = \{M_1, M_2, \ldots, M_m\}$	Set of machines
m	Number of machines
i	Index of machines ($1 \le i \le m$)
t_{ij}	Processing time of ith operation of task J_j (on machine M_i)
$\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_n\}$	A solution (sequence of n tasks) to the problem
$C(\pi_i, j)$	Completion time of task π_j on machine M_i
$T(O_k)$	Completion time of order O_k

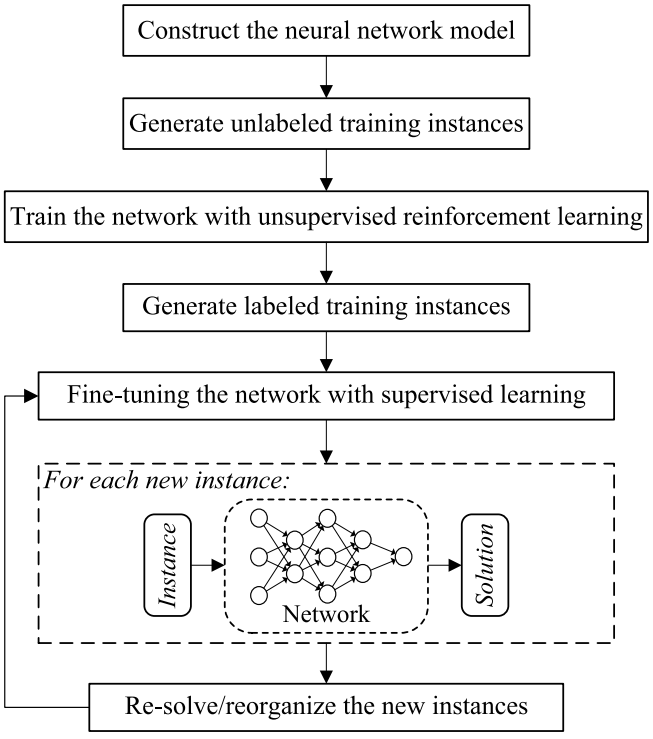


Fig. 1. Basic flow to apply the machine learning approach to solve the production task scheduling problem.

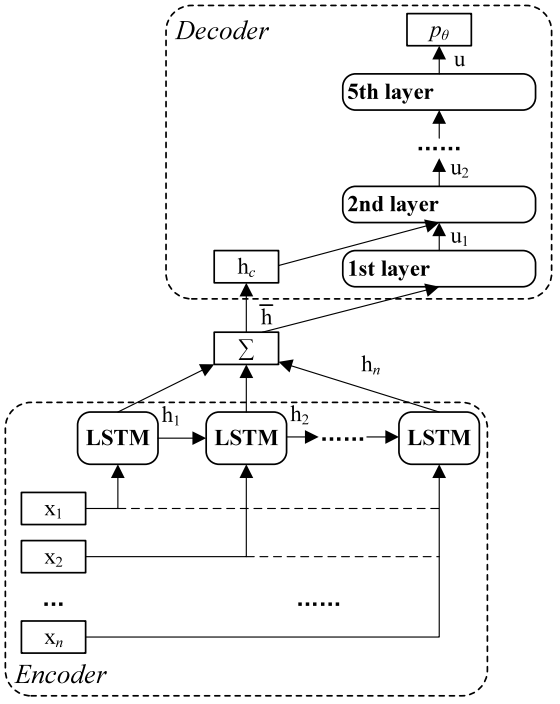


Fig. 2. Architecture of the neural network scheduler.

As a result, the encoder produces an aggregated embedding of all inputs as the mean of n hidden states:

$$\bar{\mathbf{h}} = \frac{1}{n}\sum_{j=1}^{n} \mathbf{h}_j \tag{8}$$

The decoder also performs n decoding steps, each making a decision on which task should to be processed at the next step. At each jth step, it constructs a context vector $\mathbf{h}_c$ by concatenating $\bar{\mathbf{h}}$ and the hidden state $\mathbf{h}_{j-1}$ of the previous LSTM. We use a five-layer DNN to implement the decoder. The first layer takes $\bar{\mathbf{h}}$ as input and transforms it into a n_1-dimensional hidden vector $\mathbf{u}_1$ ($n_1 < n$):

$$\mathbf{u}_1 = \mathrm{ReLU}(W_1\bar{\mathbf{h}}^{\mathrm{T}} + \mathbf{b}_1) \tag{9}$$

where W_1 is a $n_1 \times n$ weight matrix and $\mathbf{b}_1$ is a n_1-dimensional bias vector.

The second layer takes the concatenation of $\mathbf{u}_1$ and context vector $\mathbf{h}_c$ as input and transforms it into a n_2-dimensional hidden vector $\mathbf{u}_2$ ($n_2 < n_1$):

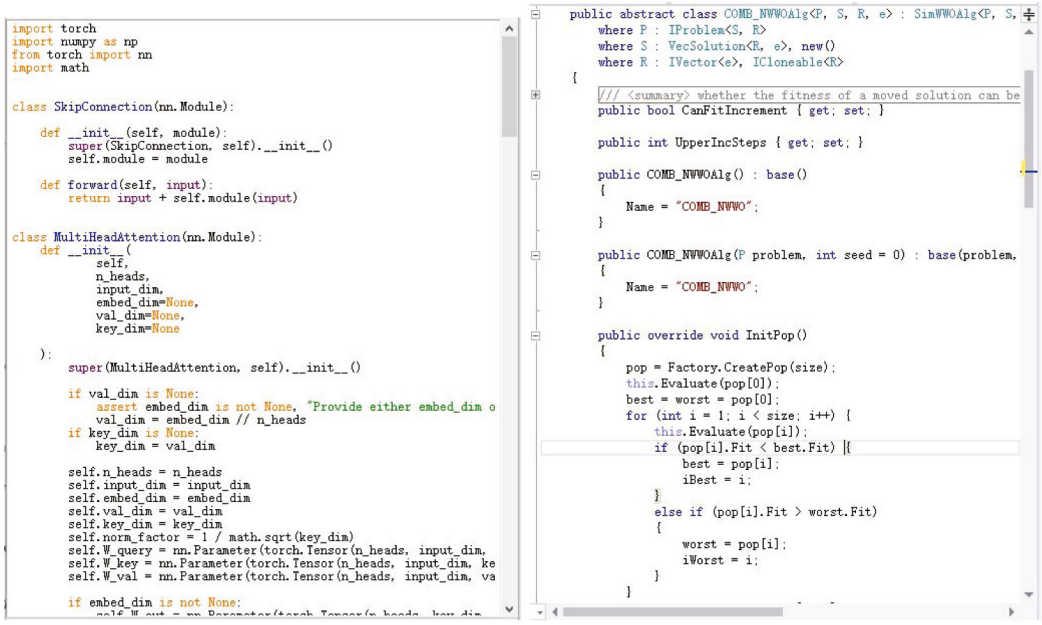
$$\mathbf{u}_2 = \mathrm{ReLU}(W_2[\mathbf{u}_1; \mathbf{h}_c]^{\mathrm{T}} + \mathbf{b}_2) \tag{10}$$

where $[\_;\_]$ denotes the horizontal concatenation of vectors, W_2 is a $n_2 \times (n_1 + 2n)$ weight matrix and $\mathbf{b}_2$ is a n_2-dimensional bias vector.

Each of the remaining layers takes the hidden state of the previous layer and transforms it into a lower-dimensional hidden vector using ReLU activation. Finally, the probability that each task x_j is selected at the tth step is calculated based on the state $\mathbf{u}$ of the top layer of the DNN:

$$p_\theta(\pi_t = x_j | \mathbf{x}, \boldsymbol{\pi}_{1:t-1}) = \frac{e^{u_j}}{\sum_{j'=1}^{n} e^{u_{j'}}} \tag{11}$$

At each step, the task that has the maximum probability is selected into the schedule.



```python
import torch
import numpy as np
from torch import nn
import math


class SkipConnection(nn.Module):

    def __init__(self, module):
        super(SkipConnection, self).__init__()
        self.module = module

    def forward(self, input):
        return input + self.module(input)


class MultiHeadAttention(nn.Module):
    def __init__(
            self,
            n_heads,
            input_dim,
            embed_dim=None,
            val_dim=None,
            key_dim=None

    ):
        super(MultiHeadAttention, self).__init__()

        if val_dim is None:
            assert embed_dim is not None, "Provide either embed_dim o
            val_dim = embed_dim // n_heads
        if key_dim is None:
            key_dim = val_dim

        self.n_heads = n_heads
        self.input_dim = input_dim
        self.embed_dim = embed_dim
        self.val_dim = val_dim
        self.key_dim = key_dim
        self.norm_factor = 1 / math.sqrt(key_dim)
        self.W_query = nn.Parameter(torch.Tensor(n_heads, input_dim,
        self.W_key = nn.Parameter(torch.Tensor(n_heads, input_dim, ke
        self.W_val = nn.Parameter(torch.Tensor(n_heads, input_dim, va

        if embed_dim is not None:
```

(a) Python codes

```
public abstract class COMB_NWWOAlg<P, S, R, e> : SimWWOAlg<P, S,
    where P : IProblem<S, R>
    where S : VecSolution<R, e>, new()
    where R : IVector<e>, ICloneable<R>
{
    /// <summary> whether the fitness of a moved solution can be
    public bool CanFitIncrement { get; set; }

    public int UpperIncSteps { get; set; }

    public COMB_NWWOAlg() : base()
    {
        Name = "COMB_NWWO";
    }

    public COMB_NWWOAlg(P problem, int seed = 0) : base(problem,
    {
        Name = "COMB_NWWO";
    }

    public override void InitPop()
    {
        pop = Factory.CreatePop(size);
        this.Evaluate(pop[0]);
        best = worst = pop[0];
        for (int i = 1; i < size; i++) {
            this.Evaluate(pop[i]);
            if (pop[i].Fit < best.Fit) {
                best = pop[i];
                iBest = i;
            }
            else if (pop[i].Fit > worst.Fit)
            {
                worst = pop[i];
                iWorst = i;
            }
        }
```

(b) Visual C# codes

Fig. 3. Snapshots of the codes of algorithmic programs.

4.2. Reinforcement learning of the neural network

A solution to the scheduling problem can be viewed as a sequence of decisions, and the decision process can be regarded as a Markov decision process [29]. According to the objective function (6), the training of the network is to minimize the loss

$$\mathcal{L}(\boldsymbol{\theta}|\mathbf{x}) = \mathbb{E}_{\boldsymbol{\pi}\sim p_\theta(\_|\mathbf{x})} f(\boldsymbol{\pi}|\mathbf{x}) \tag{12}$$

We employ the policy gradient using REINFORCE algorithm [30] with Adam optimizer [31] to train the network. The gradients of network parameters $\boldsymbol{\theta}$ are defined based on a baseline $base(\mathbf{x})$ as:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}|\mathbf{x}) = \mathbb{E}_{\boldsymbol{\pi}\sim p_\theta(\_|\mathbf{x})}\big((f(\boldsymbol{\pi}|\mathbf{x}) - base(\mathbf{x}))\nabla_{\boldsymbol{\theta}} \log p_\theta(\boldsymbol{\pi}|\mathbf{x})\big) \tag{13}$$

A good baseline reduces gradient variance and increases learning speed [16]. Here, we use both the NEH heuristic [6] and Suliman heuristic [7] to solve each instance $\mathbf{x}$, and use the better one as the $base(\mathbf{x})$.

During the training, we approximate the gradient via Monte-Carlo sampling, where B problem instances are drawn from the same distribution:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{B}\sum_{i=1}^{B}\big((f(\boldsymbol{\pi}_i|\mathbf{x}_i) - base(\mathbf{x}_i))\nabla_{\boldsymbol{\theta}} \log p_\theta(\boldsymbol{\pi}_i|\mathbf{x}_i)\big) \tag{14}$$

The pseudocode of the REINFORCE algorithm for optimizing the network parameters according to $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ is presented in Algorithm 1.

Algorithm 1: The REINFORCE algorithm.

```
Randomly initialize the network parameters θ;
for epoch = 1 to epoch_max do
    for t = 1 to T do
        for i = 1 to B do
            Sample an instance x_i from the problem distribution;
            Compute the model output π_i;
            Compute the baseline base(x_i);
        g(θ) ← (1/B) Σ_{i=1}^{B} (f(π_i) − base(x_i)) ∇_θ log p_θ(π_i|x_i));
        θ ← Adam(θ, g(θ));
return θ.
```

5. Computational results

5.1. Comparison of different baselines for the reinforcement learning algorithm

In the training phase, according to production tasks of the manufacturer during the peak of COVID-19 in China, we randomly generate 20,000 instances. The basic features of the instance distribution are as follows: $m = 5$ (the number of machines in the ZHENDE company), n follows a normal distribution $N(124, 33)$, t_{ij} follows a normal distribution $N(2.4, 1.6)$ (in hours), and d_k follows a uniform discrete distribution $\{24, 36, 48, 60, 72, 96, 120\}$ (in hours). To avoid training instances deviating too much from the distribution of the problem, any instance where the sum of all t_{ij} is larger than 1500 or less than 50 is considered as noise and is not included in the training set.

Our REINFORCE algorithm uses hybrid NEH and Suliman heuristics (denoted by NEH-Sul) as the baseline. For comparison, we also use three other baselines: the first is a greedy heuristic that sorts tasks in decreasing order of $w_k/(\sum_{i=1}^{m} t_{ij})$, and the second and the third are individual NEH heuristic and individual Suliman heuristic, respectively. For each baseline, we run 30 Monte Carlo simulations with different sequences of training instances. The maximum number of epochs for training the network is set to 100. The neural network model is implemented using Python 3.4, and the training heuristics are implemented with Microsoft Visual C# 2015 (snapshots are shown in Fig. 3 and codes can be download from http://compintell.cn/en/dataAndCode.html). The experiments are conducted on a computer with Intel Xeon 3430 CPU, 4G DDR4 Memory, and GeForce GTX 1080Ti GPU.

Fig. 4 presents the convergence curves of the four methods (averaged over the 30 Monte Carlo simulations) during the training process. The horizontal axis denotes the training epochs, and the vertical axis denotes the average objective function value

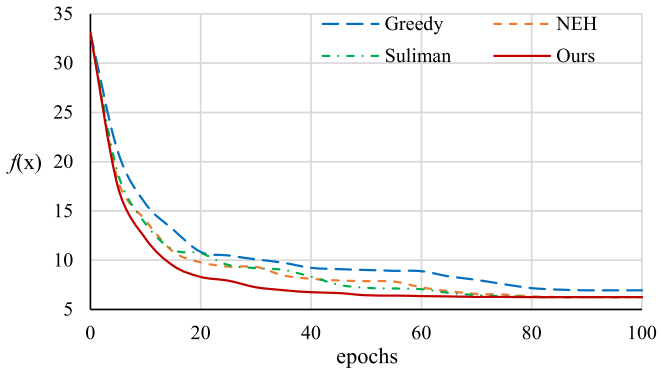


Fig. 4. Convergence curves of the four methods for training the neural network.

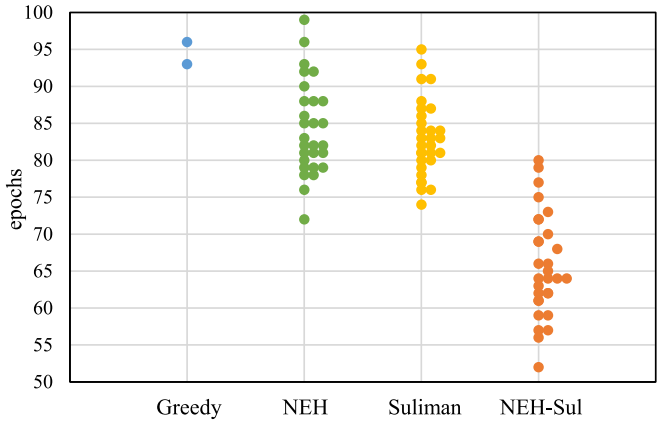


Fig. 5. Numbers of epochs of the four methods to reach the best value within 1% error.

of Eq. (6) obtained by the network. Among all simulations, the best average objective function value is approximately 6.25, and Fig. 5 presents the numbers of epochs at which the simulations converge to the best value within 1% error. The results show that, among 30 runs, the greedy method converges to the best value only twice, and in most cases it converges to local optima that are significantly worse than the best value. The individual NEH heuristic converges to the best value in 28 of the 30 runs, and the individual Suliman heuristic and our hybrid method converge to the best value in all 30 runs. In average, NEH and Suliman heuristics converge after 80∼85 epochs, while our method converges after 60∼65 epochs. This is because NEH and Suliman heuristics exhibit different performance on different instances, and using the baseline combining them can obtain solutions that are closer to the exact optima than either individual heuristic and, therefore, reduces gradient variance and increases learning speed. Moreover, as both NEH and Suliman heuristics are efficient constructive heuristics, using the hybrid baseline only incurs a slight performance overhead during the training process. The results demonstrate that, compared to the existing heuristic baselines, our method using hybrid NEH and Suliman heuristics as the baseline can significantly improve the training performance. This also provides an approach to improve reinforcement learning for neural optimization by simply combing two or more complementary baselines to a better baseline.

5.2. Comparison of scheduling performance

Next, we test the performance of the trained neural network scheduler (denoted by NNS) for solving the scheduling problem

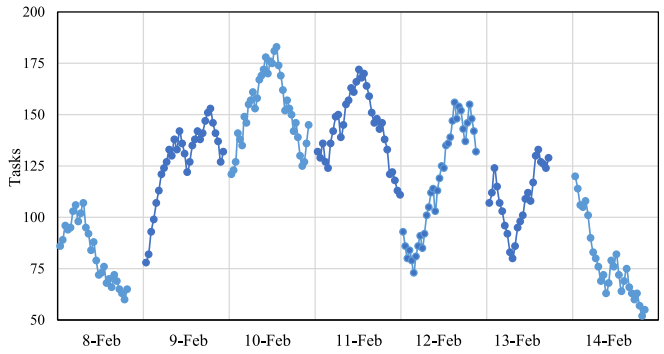


Fig. 6. Distribution of number of tasks of the real-world instances.

by comparing with the NEH heuristic, Suliman heuristic, and five state-of-the-art metaheuristic algorithms including a shuffled complex evolution algorithm (SCEA) [32], an algebraic differential evolution (ADE) algorithm [33], a teaching–learning based optimization (TLBO) algorithm [34], a biogeography-based optimization (BBO) algorithm [35,36], and a discrete water wave optimization (WWO) algorithm [15,37]. Before applying the network to real-world instances, we select 50 instances with different sizes from the training instances, use the above five metaheuristics to solve each of them, and select the best solution obtained by them as the label of the instance. The network is fine-tuned using back-propagation on the labeled instances.

We select 146 real-world instances of the manufacturer from Feb 8 to Feb 14, 2020, the peak of COVID-19 in China. For each day, we need to first solve an instance with about 50∼200 tasks; during the daytime, with the arrival of new orders, we need to reschedule the production for 20∼40 times. Fig. 6 shows the distribution of number of tasks of the instances. After each day, we also select 2∼4 scheduling/rescheduling instances, employ five metaheuristics to obtain solutions as the labels of the instances, and use the labeled instances to re-train the network. Such periodical re-training also increases the model robustness against noise.

Figs. 7–13 present the resulting objective function values obtained by the different algorithms on the instances during Feb 8 to Feb 14. For the above five stochastic metaheuristic algorithms, we perform 50 Monte Carlo simulations on each instance, and present the maximum, minimum, median, first quartile (Q1), and third quartile (Q3) of the resulting objective function values in the plots. Table 3 presents the average CPU time consumed by the algorithms to obtain the solutions; for the five metaheuristic algorithms, the stop condition is that the number of fitness evaluations reaches 100,000.

As it can be observed from the results, the Suliman heuristic typically consumes more computational time than the NEH heuristic, because the former uses a bit more complex solution construction procedure, although both heuristics have time complexities that are polynomial in instance size. The overall performance of the two heuristics are similar: the Suliman heuristic obtains better solutions on three instances, while NEH performs better on the other four instances. The computational time of our NNS model is similar to NEH (and less than Suliman): as the NNS model iteratively process input tasks and select tasks into a schedule, its time complexity is also polynomial in instance size; the time consumed to process a task by NNS is similar to the time of a construction step in NEH. Nevertheless, the solutions produced by NNS are significantly better than those of both NEH and Suliman heuristics, which demonstrates that NNS trained on the large number of instances by reinforcement learning based on the hybrid NEH and Suliman baseline can effectively learn

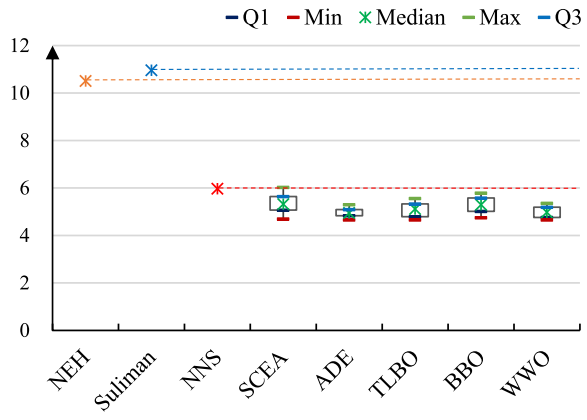


Fig. 7. Comparison of the results of the neural network scheduler, constructive heuristics, and metaheuristic algorithms on the instance of Feb 8.

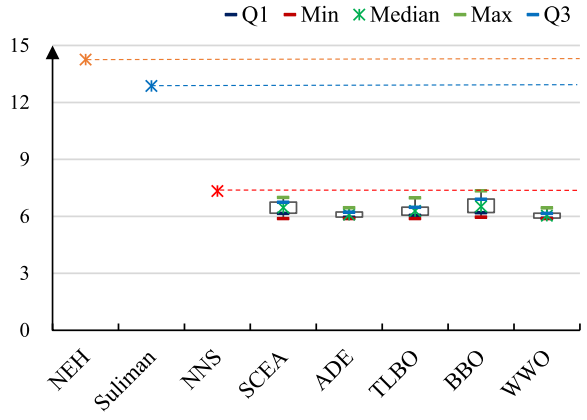


Fig. 8. Comparison of the results of the neural network scheduler, constructive heuristics, and metaheuristic algorithms on the instance of Feb 9.

the problem distribution and map new instances to high-quality solutions, and such neural network mapping exhibits significant performance advantages over the construction procedures of Suliman and NEH.

The metaheuristic algorithms are more powerful in solving the production task scheduling problem, but their performance advantages are at expense of high computational cost. On the seven problem instances, their average solution times are 500∼1500 s, significantly longer than the 1∼2 s of NNS. Nevertheless, the objective function values produced by NNS are only approximately 10%∼20% larger than the median objective function values of the metaheuristic algorithms. In some cases, the solutions of NNS are even better than the worst solutions produced by the metaheuristic algorithms (e.g., SCEA on the instance of Feb. 8 and BBO on the instance of Feb. 9). In general, compared to the state-of-the-art metaheuristics, NNS consumes about 1/1000 computational time to achieve similar performance. This is because, at each generation of the metaheuristics, the operations for evolving the solutions have time complexity polynomial in solution length (which is equivalent to instance size); as discussed above, the time complexity of NNS is also polynomial in instance size; however, the metaheuristics use approximately a thousand iterations in average. In emergency conditions, the computational time of the metaheuristics is obviously unaffordable, while the proposed NNS can produce high-quality solutions within seconds and, therefore, satisfy the requirements of emergency medical mask production.

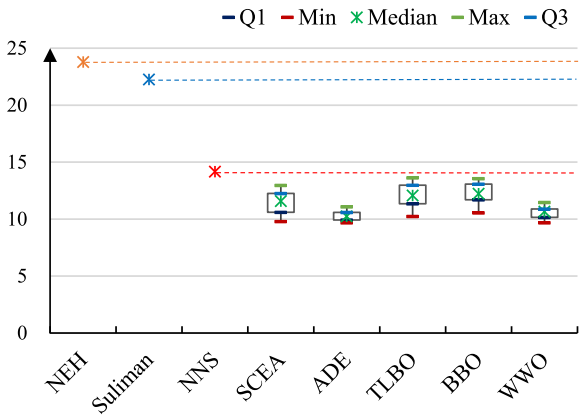


Fig. 9. Comparison of the results of the neural network scheduler, constructive heuristics, and metaheuristic algorithms on the instance of Feb 10.

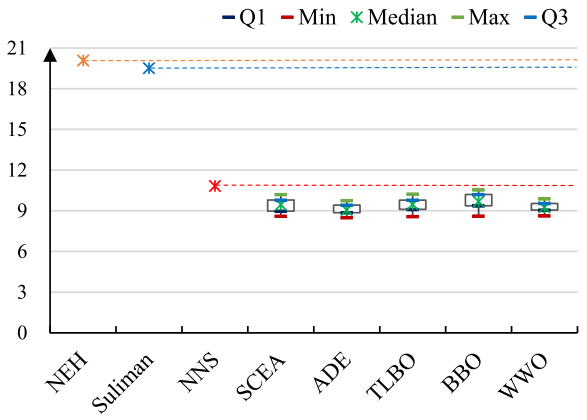


Fig. 10. Comparison of the results of the neural network scheduler, constructive heuristics, and metaheuristic algorithms on the instance of Feb 11.

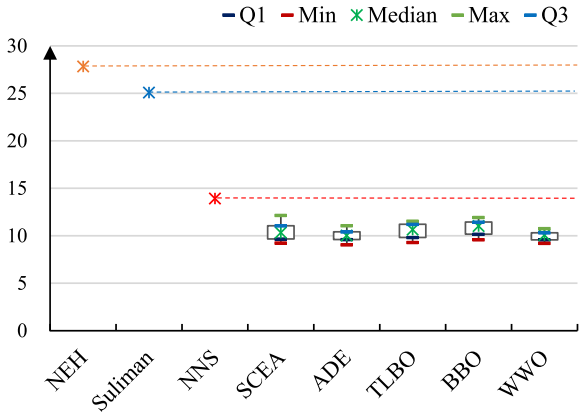


Fig. 11. Comparison of the results of the neural network scheduler, constructive heuristics, and metaheuristic algorithms on the instance of Feb 12.

6. Conclusion

In this paper, we propose a DNN with reinforcement learning for scheduling hundreds of emergency production tasks within seconds. The neural network consists of an encoder and a decoder. The encoder employs an LSTM-based RNN to sequentially parse the input production tasks, and the decoder employs a deep neural network to learn the probability distribution over these tasks. The network is trained by reinforcement learning using

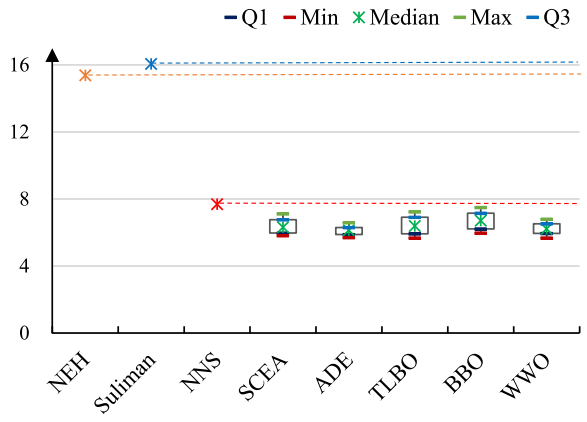


Fig. 12. Comparison of the results of the neural network scheduler, constructive heuristics, and metaheuristic algorithms on the instance of Feb 13.

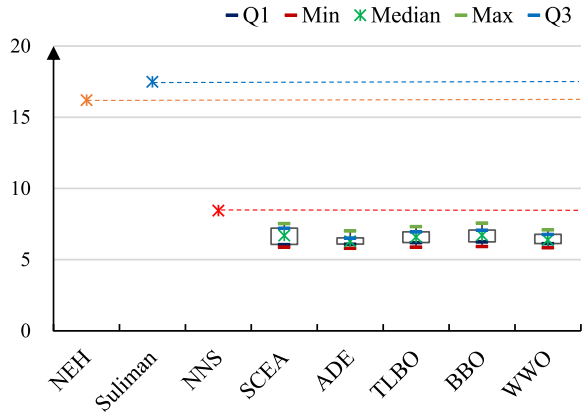


Fig. 13. Comparison of the results of the neural network scheduler, constructive heuristics, and metaheuristic algorithms on the instance of Feb 14.

Table 3
CPU time (in seconds) consumed by the neural network scheduler, constructive heuristics, and metaheuristic algorithms on the test instances.

Day	NEH	Suliman	NNS	SCEA	ADE	TLBO	BBO	WWO
Feb-8	0.63	0.93	**0.75**	545	516	525	524	520
Feb-9	0.92	2.35	**1.03**	725	690	710	718	707
Feb-10	1.71	4.59	**1.32**	1102	1027	1060	1089	1047
Feb-11	2.13	5.26	**1.57**	1210	1097	1136	1165	1121
Feb-12	2.30	5.96	**1.72**	1503	1312	1337	1422	1359
Feb-13	0.97	2.64	**1.09**	940	851	890	908	883
Feb-14	1.45	3.84	**1.20**	1013	948	970	993	956

the negative total tardiness as the reward signal. We applied the proposed neural network scheduler to a medical mask manufacturer during the peak of COVID-19 in China. The results show that the proposed approach can achieve high-quality solutions within very short computational time to satisfy the requirements of emergency production.

Emergency production scheduling is an important function that determines the efficiency of a manufacturing system in response to unexpected emergencies. Most manufacturers have purchased OR tools with exact optimization algorithms, which are suitable for only small-size instances. Many manufacturers have also equipped heuristic and metaheuristic scheduling algorithms. Constructive heuristics can produce good solutions on small-size instances and occasionally on moderate-size instances, but there is no guarantee on the optimality or optimality gap. In general, we do not encourage directly using constructive heuristics to solve real-world emergency production scheduling instances; they can be used in combined with other methods, e.g., for generating initial solutions for metaheuristics or acting baselines for neural optimization models. Metaheuristics are suitable for moderate-size instances, but are not suitable for large-size instances under emergency conditions. The proposed neural optimization method is suitable for moderate- and large-size instances, but the main disadvantage is that it should have been trained on a large number of well-distributed instances. If the emergency will last for a certain period, we can first analyze the distribution of the problem and generate sufficient instances for model training, and use the trained network in the remaining stages. Managers of manufacturers should define a decision-making process on selecting the most suitable solution methods for different instances so as to efficiently respond to emergencies.

In our study, the baseline plays a key role in reinforcement learning. The baseline used in this paper is based on two constructive heuristics, which have much room to be improved. However, better heuristics and metaheuristics often require large computational resource and are not efficient in training a large number of test instances. Currently, we are incorporating other neural network schedulers to improve the baseline. Ongoing work also includes preprocessing training instances by clustering and re-sampling to improving learning performance [38] and using metaheuristic algorithms to optimize the parameters of the deep neural network [39]. Moreover, we believe that the proposed approach can be adapted or extended to many other emergency scheduling problems, e.g., disaster relief task scheduling [40] and online unmanned aerial vehicle scheduling [41,42], in which short solution time is critical to the mission success.

Declaration of competing interest

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

Acknowledgments

This work was supported by National Natural Science Foundation of China under Grant 61872123, Zhejiang Provincial Natural Science Foundation, China under Grant LR20F030002 and LQY20F030001, and Zhejiang Provincial Emergency Project for Prevention & Treatment of New Coronavirus Pneumonia, China under Grant 2020C03126.

References

[1] M. Pinedo, Scheduling Theory, Algorithms, and Systems, second ed., Prentice Hall, New York, 2002.
[2] G.B. McMahon, P.G. Burton, Flow-shop scheduling with the branch-and-bound method, Oper. Res. 15 (3) (1967) 473–481, http://dx.doi.org/10.1287/opre.15.3.473.
[3] J.K. Karlof, W. Wang, Bilevel programming applied to the flow shop scheduling problem, Comput. Oper. Res. 23 (5) (1996) 443–451, http://dx.doi.org/10.1016/0305-0548(95)00034-8.
[4] M. Ziaee, S. Sadjadi, Mixed binary integer programming formulations for the flow shop scheduling problems, a case study: ISD projects scheduling, Appl. Math. Comput. 185 (1) (2007) 218–228, http://dx.doi.org/10.1016/j.amc.2006.06.092.
[5] C. Gicquel, L. Hege, M. Minoux, W. van Canneyt, A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints, Comput. Oper. Res. 39 (3) (2012) 629–636, http://dx.doi.org/10.1016/j.cor.2011.02.017.
[6] M. Nawaz, E.E. Enscore, I. Ham, A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem, Omega 11 (1) (1983) 91–95, http://dx.doi.org/10.1016/0305-0483(83)90088-9.
[7] S.M.A. Suliman, A two-phase heuristic approach to the permutation flow-shop scheduling problem, Int. J. Prod. Econom. 64 (1) (2000) 143–152, http://dx.doi.org/10.1016/S0925-5273(99)00053-5.

[8] Y. Zheng, J. Xue, A problem reduction based approach to discrete optimization algorithm design, Computing 88 (1–2) (2010) 31–54, http://dx.doi.org/10.1007/s00607-010-0085-0.
[9] O. Etiler, B. Toklu, M. Atak, J. Wilson, A genetic algorithm for flow shop scheduling problems, J. Oper. Res. Soc. 55 (8) (2004) 830–835, http://dx.doi.org/10.1057/palgrave.jors.2601766.
[10] G. Onwubolu, D. Davendra, Scheduling flow shops using differential evolution algorithm, European J. Oper. Res. 171 (2) (2006) 674–692, http://dx.doi.org/10.1016/j.ejor.2004.08.043.
[11] C.J. Liao, C.T. Tseng, P. Luarn, A discrete version of particle swarm optimization for flowshop scheduling problems, Comput. Oper. Res. 34 (10) (2007) 3099–3111, http://dx.doi.org/10.1016/j.cor.2005.11.017.
[12] I.H. Kuo, S.J. Horng, T.W. Kao, T.L. Lin, C.L. Lee, T. Terano, Y. Pan, An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model, Expert Syst. Appl. 36 (3) (2009) 7027–7032, http://dx.doi.org/10.1016/j.eswa.2008.08.054.
[13] J. Lin, A hybrid discrete biogeography-based optimization for the permutation flow-shop scheduling problem, Int. J. Prod. Res. 54 (16) (2016) 4805–4814, http://dx.doi.org/10.1080/00207543.2015.1094584.
[14] F. Zhao, H. Liu, Y. Zhang, W. Ma, C. Zhang, A discrete water wave optimization algorithm for no-wait flow shop scheduling problem, Expert Syst. Appl. 91 (2018) 347–363, http://dx.doi.org/10.1016/j.eswa.2017.09.028.
[15] Y.-J. Zheng, X.-Q. Lu, Y.-C. Du, Y. Xue, W.-G. Sheng, Water wave optimization for combinatorial optimization: Design strategies and applications, Appl. Soft Comput. 83 (2019) 105611, http://dx.doi.org/10.1016/j.asoc.2019.105611.
[16] W. Kool, H. van Hoof, M. Welling, Attention, learn to solve routing problems! in: International Conference on Learning Representations, 2019.
[17] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: A methodological tour d'horizon, European J. Oper. Res. (2020) http://dx.doi.org/10.1016/j.ejor.2020.07.063, (in press).
[18] J.J. Hopfield, D.W. Tank, Neural computation of decisions in optimization problems, Bio. Cybern. 52 (3) (1985) 141–152, http://dx.doi.org/10.1007/BF00339943.
[19] F.Y.-P. Simon, Takefuji, Integer linear programming neural networks for job-shop scheduling, in: International Conference on Neural Networks, vol. 2, 1988, pp. 341–348, http://dx.doi.org/10.1109/ICNN.1988.23946.
[20] G.R. Weckman, C.V. Ganduri, D.A. Koonce, A neural network job-shop scheduler, J. Intell. Manuf. 19 (2) (2008) 191–201, http://dx.doi.org/10.1007/s10845-008-0073-9.
[21] T.R. Ramanan, R. Sridharan, K.S. Shashikant, A.N. Haq, An artificial neural network based heuristic for flow shop scheduling problems, J. Intell. Manuf. 22 (2) (2011) 279–288, http://dx.doi.org/10.1007/s10845-009-0287-5.
[22] O. Vinyals, M. Fortunato, N. Jaitly, in: C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 28, Curran Associates, Inc., 2015, pp. 2692–2700.
[23] M. Nazari, A. Oroojlooy, L. Snyder, M. Takac, Reinforcement learning for solving the vehicle routing problem, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 31, Curran Associates, Inc, 2018, pp. 9839–9849.
[24] J.J.Q. Yu, W. Yu, J. Gu, Online vehicle routing with neural combinatorial optimization and deep reinforcement learning, IEEE Trans. Intell. Transp. Syst. 20 (10) (2019) 3806–3817, http://dx.doi.org/10.1109/TITS.2019.2909109.
[25] B. Peng, J. Wang, Z. Zhang, A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems, in: K. Li, W. Li, H. Wang, Y. Liu (Eds.), International Symposium on Intelligence Computation and Applications, Springer Singapore, Singapore, 2020, pp. 636–650.
[26] R. Solozabal, J. Ceberio, Takáčč, Constrained combinatorial optimization with reinforcement learning, 2020, arXiv preprint arXiv:2006.11984.
[27] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 27, Curran Associates, Inc., 2014, pp. 3104–3112.
[28] F. Gers, J. Schmidhuber, F. Cummins, Learning to forget: continual prediction with LSTM, in: International Conference on Artificial Neural Networks, Edinburgh, UK, 1999, pp. 850–855.
[29] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, 2016, arXiv preprint arXiv:1611.09940.
[30] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, in: R.S. Sutton (Ed.), Machine Learning, Springer US, Boston, MA, 1992, pp. 5–32, http://dx.doi.org/10.1007/978-1-4615-3618-5_2.
[31] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations, 2015, http://arxiv.org/abs/1412.6980.
[32] F. Zhao, J. Zhang, J. Wang, C. Zhang, A shuffled complex evolution algorithm with opposition-based learning for a permutation flow shop scheduling problem, Int. J. Comput. Integ. Manuf. 28 (11) (2015) 1220–1235, http://dx.doi.org/10.1080/0951192X.2014.961965.
[33] V. Santucci, M. Baioletti, A. Milani, Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion, IEEE Trans. Evol. Comput. 20 (5) (2016) 682–694, http://dx.doi.org/10.1109/TEVC.2015.2507785.
[34] W. Shao, D. Pi, Z. Shao, An extended teaching-learning based optimization algorithm for solving no-wait flow shop scheduling problem, Appl. Soft Comput. 61 (2017) 193–210, http://dx.doi.org/10.1016/j.asoc.2017.08.020.
[35] D. Simon, Biogeography-based optimization, IEEE Trans. Evol. Comput. 12 (6) (2008) 702–713, http://dx.doi.org/10.1109/TEVC.2008.919004.
[36] Y.-C. Du, M.-X. Zhang, C.-Y. Cai, Y.-J. Zheng, Enhanced biogeography-based optimization for flow-shop scheduling, in: J. Qiao, X. Zhao, L. Pan, X. Zuo, X. Zhang, Q. Zhang, S. Huang (Eds.), Bio-Inspired Computing: Theories and Applications, in: Commun. Comput. Inf. Sci., Springer, Singapore, 2018, pp. 295–306.
[37] Y.-J. Zheng, Water wave optimization: A new nature-inspired metaheuristic, Comput. Oper. Res. 55 (1) (2015) 1–11, http://dx.doi.org/10.1016/j.cor.2014.10.008.
[38] Y.-J. Zheng, S.-L. Yu, T.-E. Gan, J.-C. Yang, Q. Song, J. Yang, M. Karatas, Intelligent optimization of diversified community prevention of COVID-19 using traditional chinese medicine, IEEE Comput. Intell. Mag. 15 (4) (2020) http://dx.doi.org/10.1109/MCI.2020.3019899.
[39] X.-H. Zhou, M.-X. Zhang, Z.-G. Xu, C.-Y. Cai, Y.-J. Huang, Y.-J. Zheng, Shallow and deep neural network training by water wave optimization, Swarm Evol. Comput. 50 (2019) 1–13, http://dx.doi.org/10.1016/j.swevo.2019.100561.
[40] Y.-J. Zheng, H.-F. Ling, X.-L. Xu, S.-Y. Chen, Emergency scheduling of engineering rescue tasks in disaster relief operations and its application in China, Int. Trans. Oper. Res. 22 (3) (2015) 503–518, http://dx.doi.org/10.1111/itor.12148.
[41] Y. Du, M. Zhang, H. Ling, Y. Zheng, Evolutionary planning of multi-UAV search for missing tourists, IEEE Access 7 (2019) 73480–73492, http://dx.doi.org/10.1109/ACCESS.2019.2920623.
[42] Y. Zheng, Y. Du, H. Ling, W. Sheng, S. Chen, Evolutionary collaborative human-UAV search for escaped criminals, IEEE Trans. Evol. Comput. 24 (2) (2020) 217–231, http://dx.doi.org/10.1109/TEVC.2019.2925175.