Deep Reinforcement Learning for Multiobjective Optimization

Kaiwen Li, Tao Zhang, and Rui Wang

***Abstract*—This article proposes an end-to-end framework for solving multiobjective optimization problems (MOPs) using deep reinforcement learning (DRL), that we call DRL-based multiobjective optimization algorithm (DRL-MOA). The idea of decomposition is adopted to decompose the MOP into a set of scalar optimization subproblems. Then, each subproblem is modeled as a neural network. Model parameters of all the subproblems are optimized collaboratively according to a neighborhood-based parameter-transfer strategy and the DRL training algorithm. Pareto-optimal solutions can be directly obtained through the trained neural-network models. Specifically, the multiobjective traveling salesman problem (MOTSP) is solved in this article using the DRL-MOA method by modeling the subproblem as a Pointer Network. Extensive experiments have been conducted to study the DRL-MOA and various benchmark methods are compared with it. It is found that once the trained model is available, it can scale to newly encountered problems with no need for retraining the model. The solutions can be directly obtained by a simple forward calculation of the neural network; thereby, no iteration is required and the MOP can be always solved in a reasonable time. The proposed method provides a new way of solving the MOP by means of DRL. It has shown a set of new characteristics, for example, strong generalization ability and fast solving speed in comparison with the existing methods for multiobjective optimizations. The experimental results show the effectiveness and competitiveness of the proposed method in terms of model performance and running time.**

***Index Terms*—Deep reinforcement learning (DRL), multiobjective optimization, Pointer Network, traveling salesman problem.**

I. Introduction

MULTIOBJECTIVE optimization problems arise regularly in the real world where two or more objectives are required to be optimized simultaneously. Without loss of generality, a multiobjective optimization problem (MOP) can be defined as follows:

$$\min_{\mathbf{x}} \ \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x}))$$
$$\text{s.t.} \ \ \mathbf{x} \in X \tag{1}$$

where $\mathbf{f}(\mathbf{x})$ consists of M different objective functions and $X \subseteq R_D$ is the decision space. Since the M objectives usually confilict with each other, a set of tradeoff solutions called Pareto-optimal solutions is expected to be found for MOPs.

Among MOPs, various multiobjective combinatorial optimization problems have been investigated in recent years. A canonical example is the multiobjective traveling salesman problem (MOTSP), where given n cities and M cost functions to travel from city i to j, one needs to find a cyclic tour of the n cities, minimizing the M cost functions. This is an NP-hard problem even for the single-objective TSP. The best known exact method, that is, a dynamic programming algorithm, requires a complexity of $\Theta(2^n n^2)$ for single-objective TSP. It appears to be much harder for its multiobjective version. Hence, in practice, approximate algorithms are commonly used to solve MOTSPs, that is, finding near-optimal solutions.

During the last two decades, multiobjective evolutionary algorithms (MOEAs) have proven effective in dealing with MOPs since they can obtain a set of solutions in a single run due to their population-based characteristic. NSGA-II [1] and MOEA/D [2] are two of the most popular MOEAs that have been widely studied and applied in many real-world applications. The two algorithms, as well as their variants, have also been applied to solve the MOTSP (see [3]–[5]).

In addition, several handcrafted heuristics especially designed for TSP have been studied, such as the Lin–Kernighan heuristic [6] and the 2-opt local search [7]. By adopting these carefully designed tricks, a number of specialized methods have been proposed to solve MOTSP, such as the Pareto local search method (PLS) [8], multiple objective genetic local search algorithm (MOGLS) [9], and other similar variants [10]–[12]. More other methods and details can be found in this review [13].

Evolutionary algorithms and/or handcrafted heuristics have long been recognized as suitable methods to handle such problems. However, these algorithms, as iteration-based solvers, have suffered obvious limitations that have been widely discussed [13]–[15]. First, to find near-optimal solutions, especially when the dimension of problems is large, a large number of iterations are required for population updating or iterative searching, thus usually leading to a long running time for optimization. Second, once there is a slight change in the problem, for example, changing the city locations of the MOTSP, the algorithm may need to be reperformed to compute the solutions. When it comes to newly encountered

Manuscript received September 27, 2019; revised January 3, 2020; accepted February 25, 2020. Date of publication March 18, 2020; date of current version May 18, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61773390 and Grant 71571187. This article was recommended by Associate Editor H. Ishibuchi. *(Corresponding author: Rui Wang.)*

The authors are with the College of Systems Engineering, National University of Defense Technology, Changsha 410073, China, and also with the Hunan Key Laboratory of Multi-Energy System Intelligent Interconnection Technology, Changsha 410073, China (e-mail: kaiwenli_nudt@foxmail.com; zhangtao@nudt.edu.cn; ruiwangnudt@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCYB.2020.2977661.

Digital Object Identifier 10.1109/TCYB.2020.2977661

problems, or even new instances of a similar problem, the algorithm needs to be revised to obtain a good result, which is known as the *No Free Lunch theorem* [16]. Furthermore, such problem-specific methods are usually optimized for one task only.

Carefully handcrafted evolution strategies and heuristics can certainly improve the performance. However, the recent advances in machine-learning algorithms have shown their ability of replacing humans as the engineers of algorithms to solve different problems. Several years ago, most people used man-engineered features in the field of computer vision but now the deep neural networks (DNNs) have become the main techniques. While DNNs focus on making *predictions*, deep reinforcement learning (DRL) is mainly used to learn how to make *decisions*. Thereby, we believe that DRL is a possible way of learning how to solve various optimization problems automatically, thus demanding no man-engineered evolution strategies and heuristics.

In this article, we explore the possibility of using DRL to solve MOPs, MOTSP in specific, in an end-to-end manner, that is, given n cities as input, the optimal solutions can be *directly* obtained through a forward propagation of the trained network. The network model is trained through the trial-and-error process of DRL and can be viewed as a black-box heuristic or a meta-algorithm [17] with strong learned heuristics. Because of the exploring characteristic of DRL training, the obtained model can have a strong generalization ability, that is, it can solve the problems that it never saw before.

This article is originally motivated by several recent proposed neural-network-based single-objective TSP solvers. Vinyals *et al.* [18] first proposed a Pointer Network that uses the attention mechanism [19] to predict the city permutation. This model is trained in a supervised way that requires enormous TSP examples and their optimal tours as a training set. It is hard for use and the supervised training process prevents the model from obtaining better tours than the ones provided in the training set. To resolve this issue, Bello *et al.* [20] adopted an Actor–Critic DRL training algorithm to train the Point Network with no need of providing the optimal tours. Nazari *et al.* [17] simplified the Point Network model and adds dynamic elements input to extend the model to solve the vehicle routing problem (VRP). Moreover, the advanced Transformer model is employed to solve the routing problems and proves to be effective as well [21], [22].

The recent progress in solving the TSP by means of DRL is really appealing and inspiring due to its noniterative yet efficient characteristic and strong generalization ability. However, there are no such studies concerning solving MOPs (or the MOTSP in specific) by DRL-based methods.

Therefore, this article proposes to use the DRL method to deal with MOPs based on a simple but effective framework and the MOTSP is taken as a specific test problem to demonstrate its effectiveness.

The main contributions of this article are as follows.

1) This article provides a new way of solving the MOP by means of DRL. Some encouragingly new characteristics of the proposed method have been found in comparison with classical methods, for example, strong generalization ability and fast solving speed.
2) With a slight change of the problem instance, the classical methods usually need to be reconducted from scratch, which is impractical for application, especially, for large-scale problems. In contrast, our method is robust to problem changes. Once the model is trained, it can scale to problems that the algorithm never saw before in terms of the number and locations of the cities of the MOTSP.
3) The proposed method requires much lower running time than the classical methods, since the Pareto-optimal solutions can be directly obtained by a simple forward propagation of the trained networks without any population updating or iterative searching procedures.
4) Empirical studies show that the proposed method significantly outperforms the classical methods especially for large-scale MOTSPs in terms of both convergence and diversity, while requiring much less running time.

It is noted that several papers [23], [24] have introduced the concept of *Multiobjective DRL* in the field of RL. However, they mainly focus on how to apply RL to control a robot with multiple goals, such as controlling a mountain car or controlling a submarine searching for treasures, as investigated in [23]. These studies are not explicitly proposed to deal with mathematical optimization problems like (1), and thus is out of the scope of this article.

The remainder of this article is organized as follows. Section II-A introduces the general framework of the DRL-based multiobjective optimization algorithm (DRL-MOA) that describes the idea of using DRL for solving MOPs. Section II-B elaborates the detailed modeling and the training process of solving the specific MOTSP problem by means of the proposed DRL-MOA framework. Finally, the effectiveness of the method is demonstrated through experiments in Sections III and IV.

II. Deep-Reinforcement-Learning-Based Multiobjective Optimization Algorithm

In this section, we propose to solve the MOP by means of DRL based on a simple but effective framework (DRL-MOA). First, the decomposition strategy [2] is adopted to decompose the MOP into a number of subproblems. Each subproblem is modeled as a neural network. Then, model parameters of all the subproblems are optimized collaboratively according to the neighborhood-based parameter-transfer strategy and the Actor–Critic [25] training algorithm. In particular, MOTSP is taken as a specific problem to elaborate how to model and solve the MOP based on the DRL-MOA.

A. General Framework

Decomposition Strategy: Decomposition, as a simple yet efficient way to design the multiobjective optimization algorithms, has fostered a number of researches in the community, for example, cellular-based MOGA [26], MOEA/D, MOEA/DD [27], DMOEA-εC [28], and NSGA-III [29]. The idea of decomposition is also adopted as the basic framework

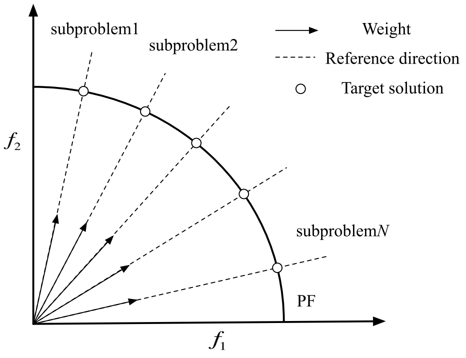


Fig. 1. Illustration of the decomposition strategy.

of the proposed DRL-MOA in this article. Specifically, the MOP, for example, the MOTSP, is explicitly decomposed into a set of scalar optimization subproblems and solved in a collaborative manner. Solving each scalar optimization problem usually leads to a Pareto-optimal solution. The desired Pareto front (PF) can be obtained when all the scalar optimization problems are solved.

In specific, the well-known weighted sum [30] approach is employed. Certainly, other scalarizing methods can also be applied, for example, the Chebyshev and the penalty-based boundary intersection (PBI) method [31], [32]. First, a set of uniformly spread weight vectors $\lambda^1, \ldots, \lambda^N$ is given, for example, $(1,0), (0.9, 0.1), \ldots, (0,1)$ for a biobjective problem, as shown in Fig. 1. Here, $\lambda^j = (\lambda_1^j, \ldots, \lambda_M^j)^T$, where M represents the number of objectives. Thus, the original MOP is converted into N scalar optimization subproblems by the weighted sum approach. The objective function of the jth subproblem is shown as follows:

$$\text{minimize}\, g^{ws}\left(x|\lambda_i^j\right) = \sum_{i=1}^{M} \lambda_i^j f_i(x). \tag{2}$$

Therefore, the PF can be formed by the solutions obtained by solving all the N subproblems.

Neighborhood-Based Parameter-Transfer Strategy: To solve each subproblem by means of DRL, the subproblem is modeled as a neural network. Then, the N scalar optimization subproblems are solved in a collaborative manner by the neighborhood-based parameter-transfer strategy, which is introduced as follows.

According to (2), it is observed that two neighboring subproblems could have very close optimal solutions [2] as their weight vectors are adjacent. Thus, a subproblem can be solved and assisted by the knowledge of its neighboring subproblems.

Specifically, as the subproblem in this article is modeled as a neural network, the parameters of the network model of the $(i-1)$th subproblem can be expressed as $[\boldsymbol{\omega}_{\boldsymbol{\lambda}^{i-1}}, \mathbf{b}_{\boldsymbol{\lambda}^{i-1}}]$. Here, $[\boldsymbol{\omega}^*, \mathbf{b}^*]$ represents the parameters of the neural-network model that have been optimized already and $[\boldsymbol{\omega}, \mathbf{b}]$ represents the parameters that are not optimized yet. Assume that the $(i-1)$th subproblem has been solved, that is, its network parameters have been optimized to its near optimum. Then, the best network parameters $[\boldsymbol{\omega}^*_{\boldsymbol{\lambda}^{i-1}}, \mathbf{b}^*_{\boldsymbol{\lambda}^{i-1}}]$ obtained in the $(i-1)$th subproblem are set as the starting point for the network training in the ith subproblem. Briefly, the network parameters are transferred from the previous subproblem to the next subproblem in a sequence, as depicted in Fig. 2. The neighborhood-based parameter-transfer strategy makes it possible for the training of the DRL-MOA model; otherwise a tremendous amount of time is required for training the N subproblems.

Algorithm 1 General Framework of DRL-MOA

```
Input: The model of the subproblem $\mathcal{M} = [\boldsymbol{w}, \mathbf{b}]$, weight vectors $\boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^N$
Output: The optimal model $\mathcal{M}^* = [\boldsymbol{w}^*, \mathbf{b}^*]$
[$\boldsymbol{\omega}_{\boldsymbol{\lambda}^1}, \mathbf{b}_{\boldsymbol{\lambda}^1}$] ← Random_Initialize
for $i \leftarrow 1 : N$ do
    if $i == 1$ then
        [$\boldsymbol{\omega}^*_{\boldsymbol{\lambda}^1}, \mathbf{b}^*_{\boldsymbol{\lambda}^1}$] ← Actor_Critic([$\boldsymbol{\omega}_{\boldsymbol{\lambda}^1}, \mathbf{b}_{\boldsymbol{\lambda}^1}$], $g^{ws}(\boldsymbol{\lambda}^1)$)
    else
        [$\boldsymbol{\omega}_{\boldsymbol{\lambda}^i}, \mathbf{b}_{\boldsymbol{\lambda}^i}$] ← [$\boldsymbol{\omega}^*_{\boldsymbol{\lambda}^{i-1}}, \mathbf{b}^*_{\boldsymbol{\lambda}^{i-1}}$]
        [$\boldsymbol{\omega}^*_{\boldsymbol{\lambda}^i}, \mathbf{b}^*_{\boldsymbol{\lambda}^i}$] ← Actor_Critic([$\boldsymbol{\omega}_{\boldsymbol{\lambda}^i}, \mathbf{b}_{\boldsymbol{\lambda}^i}$], $g^{ws}(\boldsymbol{\lambda}^i)$)
    end if
end for
return [$\boldsymbol{w}^*, \mathbf{b}^*$]
Given inputs of the MOP, the PF can be directly calculated by [$\boldsymbol{w}^*, \mathbf{b}^*$].
```

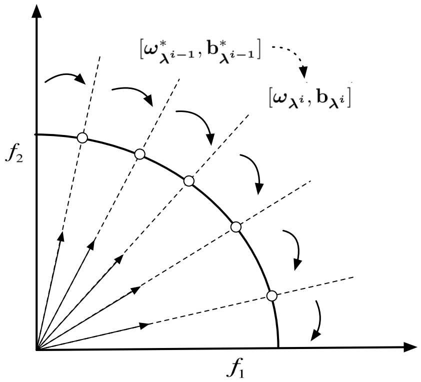

$[\boldsymbol{\omega}^*_{\boldsymbol{\lambda}^{i-1}}, \mathbf{b}^*_{\boldsymbol{\lambda}^{i-1}}]$
$[\boldsymbol{\omega}_{\boldsymbol{\lambda}^i}, \mathbf{b}_{\boldsymbol{\lambda}^i}]$
f_2
f_1

Fig. 2. Illustration of the parameter-transfer strategy.

Each subproblem is modeled and solved by the DRL algorithm and all subproblems can be solved in sequence by transferring the network weights. Thus, the PF can be finally approximated according to the obtained model. Employing the decomposition in conjunction with the neighborhood-based parameter-transfer strategy, the general framework of DRL-MOA is presented in Algorithm 1.

One obvious advantage of the DRL-MOA is its modularity and simplicity for use. For example, the MOTSP can be solved by integrating any of the recently proposed novel DRL-based TSP solvers [17], [21] into the DRL-MOA framework. Also, other problems, such as VRP and Knapsack problem, can be easily handled with the DRL-MOA framework by simply replacing the model of the subproblem. Moreover, once the trained model is available, the PF can be directly obtained by a simple forward propagation of the model.

The proposed DRL-MOA acts as an outer loop. The next issue is how to model and solve the decomposed scalar subproblems. Therefore, we take the MOTSP as a specific example and introduce how to model and solve the subproblem of MOTSP in the next section.

B. Modeling the Subproblem of MOTSP

To solve the MOTSP, we first decompose the MOTSP into a set of subproblems and solve each one collaboratively based on the foregoing DRL-MOA framework. Each subproblem is modeled and solved by means of DRL. This section introduces how to model the subproblem of MOTSP in a neural-network manner. Here, a modified Pointer Network similar to [17] is used to model the subproblem and the Actor–Critic algorithm is used for training.

1) Formulation of MOTSP: We recall the formulation of an MOTSP. One needs to find a tour of n cities, that is, a cyclic permutation ρ, to minimize M different cost functions simultaneously

$$\min z_k(\rho) = \sum_{i=1}^{n-1} c^k_{\rho(i),\rho(i+1)} + c^k_{\rho(n),\rho(1)}, \quad k = 1, \ldots, M \tag{3}$$

where $c^k_{\rho(i),\rho(i+1)}$ is the kth cost of traveling from city $\rho(i)$ to $\rho(i+1)$. The cost functions may, for example, correspond to tour length, safety index, or tourist attractiveness in practical applications.

2) Model: In this part, the above problem is modeled using a modified Pointer Network [17].

First, the input and output structures of the network model are introduced: let the given set of inputs be $X \doteq \{s^i, i = 1, \ldots, n\}$, where n is the number of cities. Each s^i is represented by a tuple $\{s^i = (s^i_1, \ldots, s^i_M)\}$. s^i_j is the attribute of the ith city that is used to calculate the jth cost function. For instance, $s^i_1 = (x_i, y_i)$ represents the x-coordinate and y-coordinate of the ith city and is used to calculate the distance between two cities. Taking a biobjective TSP as an example where both the two cost functions are defined by the Euclidean distance [13], the input structure is shown in Fig. 3. The input is 4-D and consists of total $4 \times n$ values. Moreover, the output of the model is a permutation of the cities $Y = \{\rho_1, \ldots, \rho_n\}$.

To map input X to output Y, the probability chain rule is used

$$P(Y|X) = \prod_{t=1}^{n} P(\rho_{t+1}|\rho_1, \ldots, \rho_t, X_t). \tag{4}$$

First, an arbitrary city is selected as ρ_1. At each decoding step $t = 1, 2, \ldots,$ we choose ρ_{t+1} from the available cities X_t. The available cities X_t are updated every time a city is visited. In a nutshell, (4) provides the probability of selecting the next city according to $\rho_1, \ldots, \rho_t$, that is, the already visited cities.

Then, a modified Pointer Network similar to [17] is used to model (4). Its basic structure is the Sequence-to-Sequence model [33], a recently proposed powerful model in the field of machine translation, which maps one sequence to another. The general Sequence-to-Sequence model consists of two RNN networks, called encoder and decoder. An encoder RNN encodes the input sequence into a code vector that contains the knowledge of the input. Based on the code vector, a decoder RNN is used to decode the knowledge vector to a desired sequence. Thus, the nature of the Sequence-to-Sequence model that maps one input sequence to an output sequence is suitable for solving the TSP.

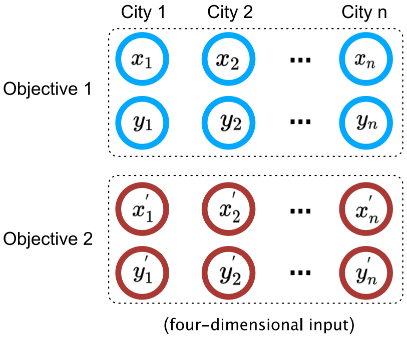


Fig. 3. Input structures of the neural-network model for solving Euclidean biobjective TSPs.

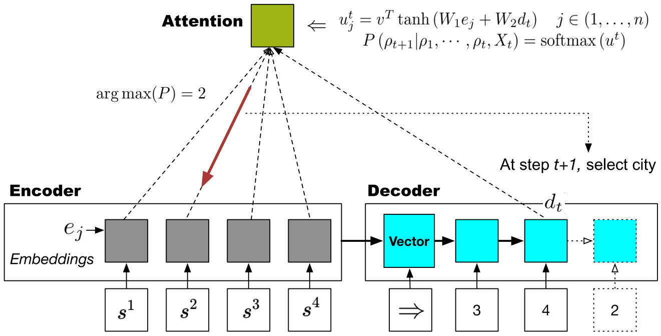

Attention
$u^t_j = v^T \tanh(W_1 e_j + W_2 d_t) \quad j \in (1, \ldots, n)$
$P(\rho_{t+1}|\rho_1, \cdots, \rho_t, X_t) = \text{softmax}(u^t)$
$\arg\max(P) = 2$
At step t+1, select city 2
Encoder
Decoder
e_j
Embeddings
d_t
Vector
s^1
s^2
s^3
s^4
⇒
3
4
2

Fig. 4. Illustration of the model structure. Attention mechanism, in conjunction with encoder and decoder, produces the probability of selecting the next city.

In this article, the architecture of the model is shown in Fig. 4, where the left part is the encoder and the right part is the decoder. The model is elaborated as follows.

Encoder: An encoder is used to condense the input sequence into a vector. Since the coordinates of the cities convey no sequential information [17] and the order of city locations in the inputs is not meaningful, RNN is not used in the encoder in this article. Instead, a simple embedding layer is used to encode the inputs to a vector which can decrease the complexity of the model and reduce the computational cost. Specifically, the 1-D convolution layer is used to encode the inputs to a *high-dimensional vector* [17] ($d_h = 128$ in this article), as shown in Fig. 4. The number of in-channels equals the dimension of the inputs. For example, the Euclidean biobjective TSP has a 4-D input as shown in Fig. 3 and thus the number of in-channels is four. The encoder finally results to an $n \times d_h$ vector, where n indicates the city number. It is noteworthy that the parameters of the 1-D convolution layer are shared among all the cities. It means that, no matter how many cities there are, each city shares the same set of parameters that encode the city information to a high-dimensional vector. Thus, the encoder is robust to the number of cities.

Decoder: The decoder is used to unfold the obtained high-dimensional vector, which stores the knowledge of inputs, into the output sequence. Different from the encoder, an RNN is required in the decoder as we need to summarize the information of previously selected cities $\rho_1, \dots, \rho_t$ so as to make the decision of ρ_{t+1}. RNN has the ability of memorizing the previous outputs. In this article, we adopt the RNN model of the gated recurrent unit (GRU) [34] that has similar performance but fewer parameters than the long short-term memory (LSTM) which is employed in the original Pointer Network in [17]. It is noted that RNN is not directly used to output the sequence. What we need is the RNN decoder hidden state d_t at decoding step t that stores the knowledge of previous steps $\rho_1, \dots, \rho_t$. Then, d_t and the encoding of the inputs $e_1, \dots, e_n$ are used together to calculate the conditional probability $P(y_{t+1}|\rho_1, \dots, \rho_t, X_t)$ over the next step of city selection. This calculation is realized by the attention mechanism. As shown in Fig. 4, to select the next city at step $t+1$, first, we obtain the hidden state d_t through decoder. In conjunction with $e_1, \dots, e_n$, the index of the next city can be calculated using the attention mechanism.

Attention Mechanism: Intuitively, the attention mechanism calculates how much every input is relevant in the next decoding step t. The most relevant one is given more *attention* and can be selected as the next visiting city. The calculation is as follows:

$$\begin{aligned} u_j^t &= v^T \tanh\big(W_1 e_j + W_2 d_t\big) \quad j \in (1, \dots, n) \\ &\times P(\rho_{t+1}|\rho_1, \dots, \rho_t, X_t) = \operatorname{softmax}\big(u^t\big) \end{aligned} \tag{5}$$

where v, W_1, and W_2 are *learnable* parameters. d_t is a key variable for calculating $P(\rho_{t+1}|\rho_1, \dots, \rho_t, X_t)$ as it stores the information of previous steps $\rho_1, \dots, \rho_t$. Then, for each city j, its u_j^t is computed by d_t and its encoder hidden state e_j, as shown in Fig. 4. The softmax operator is used to normalize $u_1^t, \dots, u_n^t$ and finally, the probability for selecting each city j at step t can be finally obtained. The greedy decoder can be used to select the next city. For example, in Fig. 4, city 2 has the largest $P(\rho_{t+1}|\rho_1, \dots, \rho_t, X_t)$ and so is selected as the next visiting city. Instead of selecting the city with the largest probability greedily, during training, the model selects the next city by sampling from the probability distribution.

3) Training Method: The model of the subproblem is trained using the well-known Actor–Critic method similar to [17] and [20]. However, as [17], [20] trains the model of single-objective TSP, the training procedure is different for the MOTSP case, as presented in Algorithm 2. Next, we briefly introduce the training procedure.

Two networks are required for training: 1) an actor network, which is exactly the Pointer Network in this article, gives the probability distribution for choosing the next action and 2) a critic network that evaluates the expected reward given a specific problem sate. The critic network employs the same architecture as the Pointer Network's encoder that maps the encoder hidden state into the critic output.

The training is conducted in an unsupervised way. During the training, we generate the MOTSP instances from distributions $\{\Phi_{\mathcal{M}_1}, \dots, \Phi_{\mathcal{M}_M}\}$. Here, $\mathcal{M}$ represents different input features of the cities, for example, the city locations or the security indices of the cities. For example, for Euclidean instances of a biobjective TSP, $\mathcal{M}_1$ and $\mathcal{M}_2$ are both city coordinates and $\Phi_{\mathcal{M}_1}$ or $\Phi_{\mathcal{M}_2}$ can be a uniform distribution of $[0, 1] \times [0, 1]$.

Algorithm 2 Actor–Critic Training Algorithm

```
Input: $\theta, \phi \leftarrow$ initialized parameters given in Algorithm 1
Output: The optimal parameters $\theta, \phi$
    for iteration $\leftarrow 1, 2, \dots$ do
2:      generate $T$ problem instances from $\{\Phi_{\mathcal{M}_1}, \dots, \Phi_{\mathcal{M}_M}\}$
        for the MOTSP.
        for $k \leftarrow 1, \dots, T$ do
4:          $t \leftarrow 0$
            while not terminated do
6:              select the next city $\rho_{t+1}^k$ according to
                $P\big(\rho_{t+1}^k|\rho_1^k, \dots, \rho_t^k, X_t^k\big)$
                Update $X_t^k$ to $X_{t+1}^k$ by leaving out the visited cities.
8:          end while
            compute the reward $R^k$
10:     end for
        $d\theta \leftarrow \frac{1}{N}\sum_{k=1}^{N}\big(R^k - V\big(X_0^k;\phi\big)\big)\nabla_\theta \log P\big(Y^k|X_0^k\big)$
12:     $d\phi \leftarrow \frac{1}{N}\sum_{k=1}^{N}\nabla_\phi\big(R^k - V\big(X_0^k;\phi\big)\big)^2$
        $\theta \leftarrow \theta + \eta d\theta$
14:     $\phi \leftarrow \phi + \eta d\phi$
    end for
```

To train the actor and critic networks with parameters θ and ϕ, N instances are sampled from $\{\Phi_{\mathcal{M}_1}, \dots, \Phi_{\mathcal{M}_M}\}$ for training. For each instance, we use the actor network with current parameters θ to produce the cyclic tour of the cities and the corresponding reward can be computed. Then, the policy gradient is computed in line 11 (refer to [35] for details of the formula derivation of policy gradient) to update the actor network. Here, $V(X_0^n;\phi)$ is the reward approximation of instance n calculated by the critic network. The critic network is then updated in line 12 by reducing the difference between the true observed rewards and the approximated rewards.

Once all of the models of the subproblems are trained, the Pareto-optimal solutions can be directly output by a simple forward propagation of the models. The time complexity of a forward calculation of encoder is $O(d_h n)$, and the time complexity of a forward calculation of decoder is $O(d_h^2 n)$, where $O(d_h^2)$ is the approximated time complexity of the RNN. Thus, the approximated time complexity of using DRL-MOA for solving the MOTSP is $O(Nnd_h^2)$, where N is the number of subproblems. As a forward propagation of the encoder–decoder neural network can be quite fast, the solutions can be always obtained within a reasonable time.

III. Experimental Setups

The proposed DRL-MOA is tested on biobjective TSPs. All experiments are conducted on a single GTX 2080Ti GPU. The code is written in Python and is publicly available[1] to reproduce the experimental results and to facilitate future studies. Meanwhile, all experiments of the compared MOEAs are conducted on the standard software platform PlatEMO[2] [36]

[1] https://github.com/kevin031060/RL_TSP_4static

[2] http://bimk.ahu.edu.cn/index.php?s=/Index/Software/index.html

which is written in MATLAB. The compared MOGLS is written in Python.[3] All of the compared algorithms are run on the Intel 16-Core i7-9800X CPU with 64-GB memory.

A. Test Instances

The considered biobjective TSP instances are described as follows [13].

Euclidean Instances: Euclidean instances are the commonly used test instances for solving MOTSP [13]. Intuitively, both the cost functions are defined by the Euclidean distance. The first cost is defined by the distance between the real coordinates of two cities i and j. The second cost of traveling from city i to city j is defined by another set of *virtual* coordinates that used to calculate another objective. Thus, the input is 4-D as shown in Fig. 3.

Mixed Instances: In order to test the ability of our model that can adapt to different input structures, Mixed instances with 3-D input are tested. Here, the first cost function is still defined by the Euclidean distance between two points that represents the real city location, which is a 2-D input with x-coordinate and y-coordinate. Moreover, the second cost of traveling from city i to j is defined by a 1-D input. This 1-D input of city i can be interpreted as the altitude of city i. Thus, the objective is to minimize the altitude variance when traveling between two cities. A smoother tour can be obtained with less altitude variance, therefore, we can reduce the fuel cost or improve the comfort of the journey. Thereby, Mixed instances have a 3-D input.

Training Set: As an unsupervised learning method, only the model input and the reward function are required during the training process, with no need for the best tours as the labels. Euclidean instances with 4-D input and Mixed instances with 3-D input are generated as the training set. They are all generated from a uniform distribution of [0, 1].

Test Set: The standard TSP test problems kroA and kroB in the TSPLIB library [37] are used to construct the Euclidean test instances kroAB100, kroAB150, and kroAB200 which are commonly used MOTSP test instances [12], [13]. kroA and kroB are two sets of different city locations and used to calculate the two Euclidean costs. For Mixed test instances, randomly generated 40-, 70-, 100-, 150-, and 200-city instances are constructed.

In this article, the model is trained on 40-city MOTSP instances and it is used to approximate the PFs of 40-, 70-, 100-, 150-, and 200-city test instances.

B. Parameter Settings of Model and Training

Most parameters of the model and training are similar to that in [17] which can solve single-objective TSPs, effectively. Specifically, the parameter settings of the network model are shown in Table I. D_{input} represents the dimension of input, that is, $D_{\text{input}} = 4$ for Euclidean biobjective TSPs. We employ a one-layer GRU RNN with the hidden size of 128 in the decoder. For the critic network, the hidden size is also set to 128.

TABLE I
PARAMETER SETTINGS OF THE MODEL. 1D-CONV MEANS THE 1-D CONVOLUTION LAYER. D_{input} REPRESENTS THE DIMENSION OF INPUT. KERNEL SIZE AND STRIDE ARE ESSENTIAL PARAMETERS OF THE 1-D CONVOLUTION LAYER

Actor network(Pointer Network)	
Encoder:	1D-Conv(D_{input}, 128, kernel size=1, stride=1)
Decoder:	GRU(hidden size=128, number of layer=1)
	Attention(No hyper parameters)
Critic network	
1D-Conv(D_{input}, 128, kernel size=1, stride =1)	
1D-Conv(128, 20, kernel size=1, stride =1)	
1D-Conv(20, 20, kernel size=1, stride =1)	
1D-Conv(20, 1, kernel size=1, stride =1)	

We train both of the actor and critic networks using the Adam optimizer [38] with the learning rate η of 0.0001 and the batch size of 200. The *Xavier* initialization method [39] is used to initialize the weights for the first subproblem. Weights for the following subproblems are generated by the introduced neighborhood-based parameter-transfer strategy.

In addition, different size of generated instances is required for training different types of models. As compared with the Mixed MOTSP problem, the model of the Euclidean MOTSP problem requires more weights to be optimized because its dimension of input is larger, thus requiring more training instances in each iteration. In this article, we generate 500 000 instances to train the Euclidean biobjective TSP and 120 000 instances to train the Mixed one. All the problem instances are generated from a uniform distribution of [0, 1] and used for training for five epochs. It costs about 3 h to train the Mixed instances and 7 h to train the Euclidean instances. Once the model is trained, it can be used to directly output the PFs.

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

In this section, DRL-MOA is compared with classical MOEAs of NSGA-II and MOEA/D on different MOTSP instances. The maximum number of iteration for NSGA-II and MOEA/D is set to 500, 1000, 2000, and 4000, respectively. The population size is set to 100 for NSGA-II and MOEA/D. The number of subproblems for DRL-MOA is set to 100 as well. The Tchebycheff approach, which we found would perform better on MOTSP, is used for MOEA/D. In addition, only the nondominated solutions are reserved in the final PF.

A. Results on Mixed-Type Biobjective TSP

We first test the model that is trained on 40-city Mixed-type biobjective TSP instances. The model is then used to approximate the PF of 40-, 70-, 100-, 150-, and 200-city instances.

The performance of the PFs obtained by all the compared algorithms on various instances are shown in Figs. 5–8. It is observed that the trained model can efficiently scale to biobjective TSP with different number of cities. Although the model

[3] https://github.com/kevin031060/Genetic_Local_Search_TSP

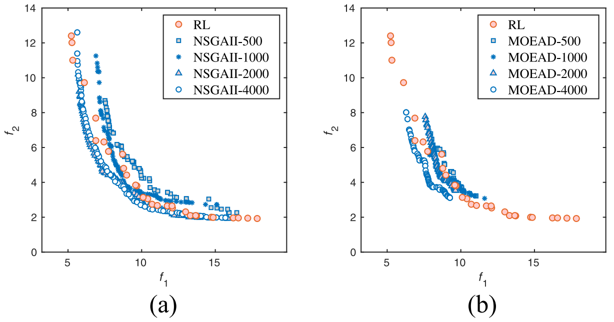


Fig. 5. Randomly generated 40-city Mixed biobjective TSP problem instance: the PF obtained using our method (trained using 40-city instances) in comparison with NSGA-II and MOEA/D. 500, 1000, 2000, and 4000 iterations are applied, respectively. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.

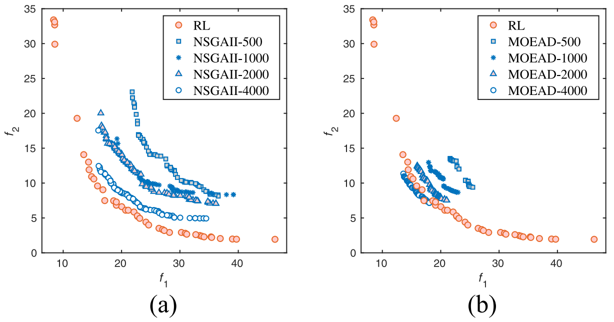


Fig. 6. Randomly generated 100-city Mixed biobjective TSP problem instance: the PF obtained using our method (trained using 40-city instances) in comparison with NSGA-II and MOEA/D. 500, 1000, 2000, and 4000 iterations are applied, respectively. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.

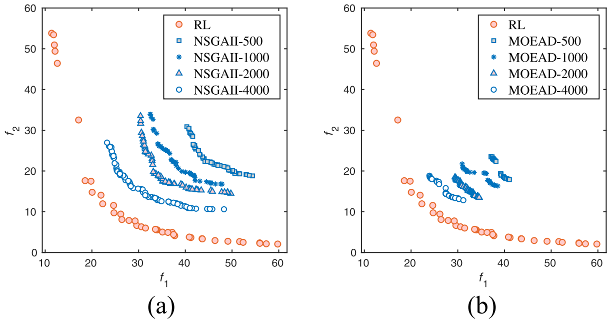


Fig. 7. Randomly generated 150-city Mixed biobjective TSP problem instance: the PF obtained using our method (trained using 40-city instances) in comparison with NSGA-II and MOEA/D. 500, 1000, 2000, and 4000 iterations are applied, respectively. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.

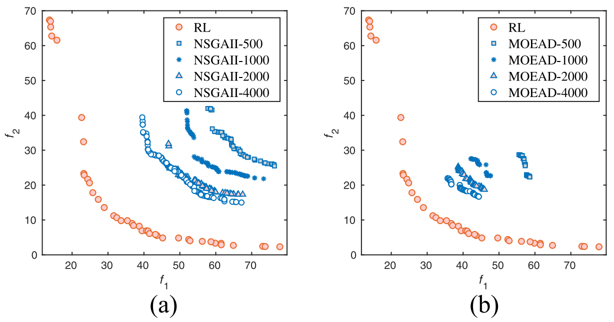


Fig. 8. Randomly generated 200-city Mixed biobjective TSP problem instance: the PF obtained using our method (trained using 40-city instances) in comparison with NSGA-II and MOEA/D. 500, 1000, 2000, and 4000 iterations are applied, respectively. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.

is obtained by training on the 40-city instances, it can still exhibit good performances on the 70-, 100-, 150-, and 200-city instances. Moreover, the performance indicator of hypervolume (HV) and the running time that are obtained based on five runs are also listed in Table II.

As shown in Fig. 5, all of the compared algorithms can work well for the small-scale problems, for example, 40-city instances. By increasing the number of iterations, NSGA-II and MOEA/D even show a better ability of convergence. However, a large number of iterations can lead to a large amount of computing time. For example, 4000 iterations cost 130.2 s for MOEA/D and 28.3 s for NSGA-II while our method just requires 2.7 s.

As can be seen in Figs. 6–8, as the number of cities increases, the competitors of NSGA-II and MOEA/D struggle to converge while the DRL-MOA exhibits a much better ability of convergence.

For 100-city instances in Fig. 6, MOEA/D shows a slightly better performance in terms of convergence than other methods by running 4000 iterations with 140.3 s. However, the diversity of solutions found by our method is much better than MOEA/D.

For 150- and 200-city instances as depicted in Figs. 7 and 8, NSGA-II and MOEA/D exhibit an obviously inferior performance than our method in terms of both the convergence and diversity. Even though the competitors are conducted for 4000 iterations, which is a pretty large number of iterations, DRL-MOA still shows a far better performance than them.

In addition, the DRL-MOA achieves the best HV compared to other algorithms, as shown in Table II. Also, its running time is much lower in comparison with the competitors. Overall, the experimental results clearly indicate the effectiveness of DRL-MOA on solving the large-scale biobjective TSPs. The *brain* of the trained model has learned how to select the next city given the city information and the selected cities. Thus, it does not suffer the deterioration of performance with the increasing number of cities. In contrast, NSGA-II and MOEA/D fail to converge within a reasonable computing time for large-scale biobjective TSPs. In addition, the PF obtained by the DRL-MOA method shows a significantly better diversity as compared with NSGA-II and MOEA/D whose PF has a much smaller spread.

B. *Results on Euclidean-Type Biobjective TSP*

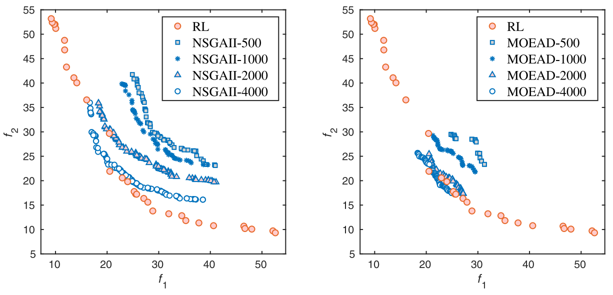
We then test the model on Euclidean-type instances. The DRL-MOA model is trained on 40-city instances and applied to approximate the PF of 40-, 70-, 100-, 150-, and 200-city instances. For 100-, 150-, and 200-city problems, we adopt the commonly used kroAB100, kroAB150, and kroAB200 instances [13]. The HV indicator and computing time are shown in Table III.

TABLE II
HV VALUES OBTAINED BY DRL-MOA, NSGA-II, AND MOEA/D. INSTANCES OF 40-, 70-, 100-, 150-, AND 200-CITY MIXED-TYPE BIOBJECTIVE TSP ARE TEST. THE RUNNING TIME IS LISTED. THE BEST HV IS MARKED IN GRAY BACKGROUND AND THE LONGEST RUNNING TIME IS MARKED BOLD

	40-city		70-city		100-city		150-city		200-city	
	HV	Time/s	HV	Time/s	HV	Time/s	HV	Time/s	HV	Time/s
NSGAII-500	1282	4.1	3866	4.2	7186	4.6	15158	5.7	26246	6.5
NSGAII-1000	1345	7.0	4042	9.6	7717	8.7	16313	12.6	27557	11.9
NSGAII-2000	1366	13.3	4146	16.8	8218	16.3	17283	21.4	29206	23.3
NSGAII-4000	1404	28.3	4434	32.7	8597	33.2	18267	40.5	31647	51.2
MOEA/D-500	1251	17.0	3878	17.7	7367	18.5	15796	20.5	26548	21.8
MOEA/D-1000	1305	34.5	4048	35.2	7796	35.9	16838	40.6	28851	41.9
MOEA/D-2000	1324	65.2	4166	68.5	8261	73.2	17833	79.4	30785	85.5
MOEA/D-4000	1346	**130.2**	4235	**136.0**	8471	**145.2**	18644	**157.6**	32642	**169.2**
DRL-MOA	1398	2.7	4668	4.7	9647	6.6	22386	10.1	40354	12.9

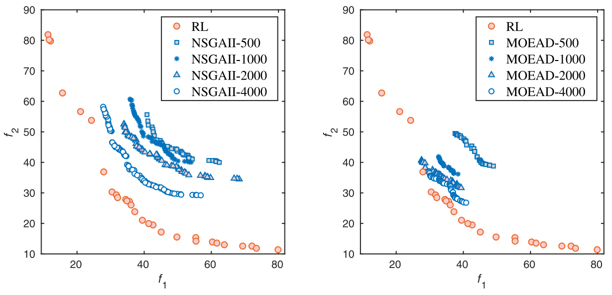
TABLE III
HV VALUES OBTAINED BY DRL-MOA, NSGA-II, AND MOEA/D. INSTANCES OF 40-, 70-, 100-, 150-, AND 200-CITY EUCLIDEAN-TYPE BIOBJECTIVE TSP ARE TEST. THE RUNNING TIME IS LISTED. THE BEST HV IS MARKED IN GRAY BACKGROUND AND THE LONGEST RUNNING TIME IS MARKED BOLD

	40-city		70-city		100-city		150-city		200-city	
	HV	Time/s	HV	Time/s	HV	Time/s	HV	Time/s	HV	Time/s
NSGAII-500	1498	3.8	4446	4.1	8738	4.3	18487	5.2	31430	5.9
NSGAII-1000	1547	7.2	4643	7.7	9119	8.1	19491	9.6	33424	11.0
NSGAII-2000	1587	13.4	4798	15.7	9577	15.6	20116	20.1	35261	23.3
NSGAII-4000	1596	26.7	4874	29.5	9816	31.8	21395	40.2	36375	54.1
MOEA/D-500	1485	16.4	4438	17.1	8851	18.5	18941	20.3	32540	21.2
MOEA/D-1000	1494	33.6	4576	34.3	9256	36.5	19897	39.7	34842	42.4
MOEA/D-2000	1525	65.2	4703	69.5	9594	71.7	20723	78.6	36253	84.8
MOEA/D-4000	1512	**130.3**	4781	**135.2**	9778	**141.7**	21522	**156.9**	37687	**168.2**
DRL-MOA	1603	2.6	5150	4.5	10773	6.3	24567	9.4	44110	12.9



(a) (b)

Fig. 9. KroAB100 Euclidean biobjective TSP problem instance: the PF obtained using our method (trained using 40-city instances) in comparison with NSGA-II and MOEA/D. 500, 1000, 2000, and 4000 iterations are applied, respectively. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.



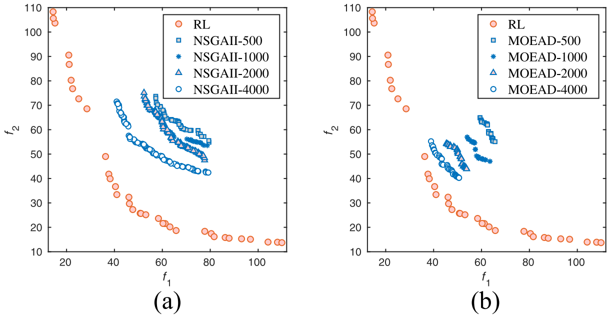
(a) (b)

Fig. 10. KroAB150 Euclidean biobjective TSP problem instance: the PF obtained using our method (trained using 40-city instances) in comparison with NSGA-II and MOEA/D. 500, 1000, 2000, and 4000 iterations are applied, respectively. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.

Figs. 9–11 show the experimental results on kroAB100, kroAB150, and kroAB200 instances. For the kroAB100 instance, by increasing the number of iterations to 4000, NSGA-II, MOEA/D, and DRL-MOA achieve a similar level of convergence while MOEA/D performs slightly better. However, MOEA/D performs the worst in terms of diversity with all solutions crowded in a small region and its running time is not acceptable.

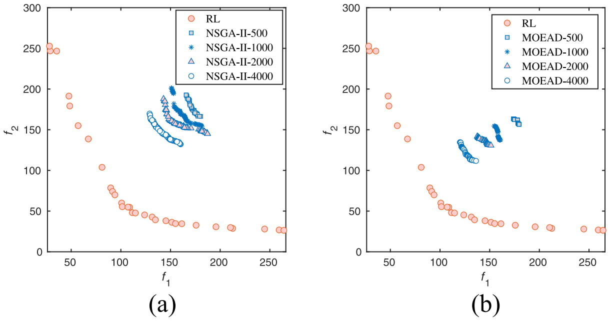
When the number of cities increases to 150 and 200, DRL-MOA significantly outperforms the competitors in terms of both convergence and diversity, as shown in Figs. 10 and 11. Even though 4000 iterations are conducted for NSGA-II and MOEA/D, there is still an obvious gap of performance between the two methods and the DRL-MOA.

In terms of the HV indicator as demonstrated in Table III, DRL-MOA performs the best in all instances. The running time of DRL-MOA is much lower than the compared MOEAs. Increasing the number of iterations for MOEA/D and NSGA-II can certainly improve the performance but would result in a large amount of computing time. It requires more than 150 s for MOEA/D to reach an acceptable level of convergence. The computing time of NSGA-II is less, approximately 30 s, for running 4000 iterations. However, the performance for NSGA-II is always the worst among the compared methods.



(a) (b)

Fig. 11. KroAB200 Euclidean biobjective TSP problem instance: the PF obtained using our method (trained using 40-city instances) in comparison with NSGA-II and MOEA/D. 500, 1000, 2000, and 4000 iterations are applied, respectively. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.



(a) (b)

Fig. 12. Randomly generated 500-city Euclidean biobjective TSP problem instance: the PF obtained using our method (trained using 40-city instances) in comparison with NSGA-II and MOEA/D. 500, 1000, 2000, and 4000 iterations are applied, respectively. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.

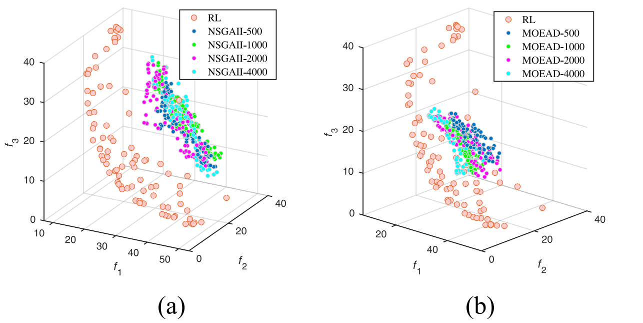
We further try to evaluate the performance of the model on 500-city instances. The model is still the one that is trained on 40-city instances and it is used to approximate the PF of a 500-city instance. The results are shown in Fig. 12. It is observed that DRL-MOA significantly outperforms the competitors and the performance gap is especially larger than that on smaller-scale problems.

C. Extension to MOTSPs With More Objectives

In this section, the efficiency of our method is further evaluated on the 3- and 5-objective TSPs. The model is still trained on 40-city instances and it is used to approximate the PF of 100- and 200-city instances. The 3-objective TSP instances are constructed by combining two 2-D inputs and a 1-D input similar to the Mixed-type instances. The 5-objective TSP instances are constructed in the same way with two 2-D inputs and three 1-D input.

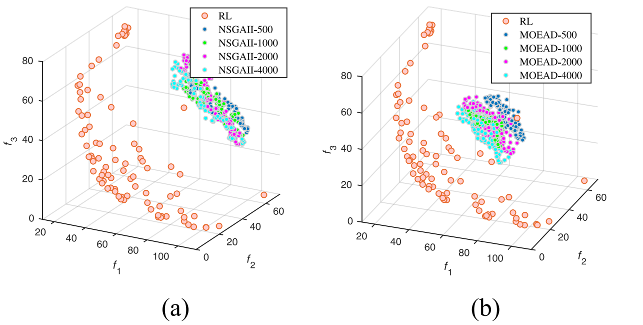
The results of the experiments on the 3-objective TSP are visualized in Figs. 13 and 14. It can be observed that DRL-MOA significantly outperforms the classical MOEAs on all of the 100- and 200-city instances. Moreover, DRL-MOA performs clearly better on the 200-city instance than on the 100-city instance while the competitors struggle to converge.

In addition, the results of the HV values on the 3- and 5-objective TSP instances that are obtained based on five runs are presented in Table IV. It can be seen that the DRL-MOA outperforms NSGA-II and MOEA/D in all instances. NSGA-II is still the least effective method.



(a) (b)

Fig. 13. PFs obtained by DRL-MOA, NSGA-II, and MOEA/D on a randomly generated 3-objective 100-city TSP instance. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.



(a) (b)

Fig. 14. PFs obtained by DRL-MOA, NSGA-II, and MOEA/D on a randomly generated 3-objective 200-city TSP instance. (a) DRL-MOA and NSGA-II. (b) DRL-MOA and MOEA/D.

TABLE IV

HV VALUES OBTAINED BY DRL-MOA, NSGA-II, AND MOEA/D ON 3- AND 5-OBJECTIVE TSP INSTANCES. THE BEST HV IS MARKED IN GRAY BACKGROUND

	3-objective TSP		5-objective TSP	
	100-city	200-city	100-city	200-city
NSGAII-500	7.63E+05	5.09E+06	5.00E+09	1.31E+11
NSGAII-1000	7.72E+05	5.41E+06	5.19E+09	1.42E+11
NSGAII-2000	8.25E+05	5.56E+06	5.51E+09	1.57E+11
NSGAII-4000	8.53E+05	5.98E+06	6.12E+09	1.64E+11
MOEA/D-500	7.74E+05	5.63E+06	5.55E+09	1.53E+11
MOEA/D-1000	8.17E+05	6.02E+06	6.19E+09	1.55E+11
MOEA/D-2000	8.58E+05	6.26E+06	6.39E+09	1.68E+11
MOEA/D-4000	8.82E+05	6.49E+06	7.15E+09	1.78E+11
RL-MOA	1.13E+06	9.42E+06	1.19E+10	4.06E+11

D. Comparisons With Local-Search-Based Methods

In this section, DRL-MOA is compared with the local-search-based method. Ishibuchi and Murata first proposed a MOGLS [40] for multiobjective combinatorial optimization. It is further improved and specialized to solve the MOTSP in [41], which significantly outperforms the original MOGLS. In this section, the DRL-MOA is compared with the improved MOGLS.

Note that local search has been widely developed and various local-search-based methods that use a number of specialized techniques to improve effectiveness have been

TABLE V
HV VALUES OF THE PFS OBTAINED BY MOGLS-100, MOGLS-200, MOGLS-300, DRL-MOA, AND DRL-MOA + LS. THE BEST AND THE SECOND BEST HVS ARE MARKED IN GRAY AND LIGHT GRAY BACKGROUND. THE LONGEST COMPUTATIONAL TIME IS MARKED BOLD

Instances		100-city		200-city	
		HV	Time/s	HV	Time/s
Mixed	MOGLS-100	1848	143.3	5800	394.6
	MOGLS-200	1986	254.9	6516	738.8
	MOGLS-300	2037	**349.8**	6794	**1073.7**
	DRL-MOA	2022	6.6	6911	12.9
	DRL-MOA+LS	2073	12.7	6988	23.2
		HV	Time/s	HV	Time/s
Euclidean	MOGLS-100	3127	138.1	13799	374.8
	MOGLS-200	3362	246.3	15093	697.1
	MOGLS-300	3450	**348.1**	15716	**1005.6**
	DRL-MOA	3342	6.4	15750	12.9
	DRL-MOA+LS	3474	12.4	16117	24.1

proposed these years. However, the goal of our method is not to outperform a nonlearned, specialized MOTSP algorithm. Rather, we show the fast solving speed and high generalization ability of our method via the combination of DRL and multiobjective optimization. Thus, we did not consider more other local-search-based methods in this article.

As no source code is found, the improved MOGLS is implemented by ourselves strictly according to [41]. The algorithm is written in Python and run in the same machine to make fair comparisons and the code is publicly available.[4]

The parameters, for example, the size of the temporary population and the number of the initial solution are consistent with the settings in [41]. The local search uses a standard 2-opt algorithm, which is terminated if a prespecified number of iterations N_{LS} is completed. N_{LS} is used to control the balance of computational time and model performance [40]. The improved MOGLS with $N_{LS} = 100, 200, 300$ is used for comparisons. It should be noted that improving N_{LS} or repeating the local search until no better solution is found might be able to further improve the performance of MOGLS. But it would be quite time consuming and not experimented in this article.

Since the local search can further improve the quality of solutions obtained by DRL as reported in [22], we thus use a simple 2-opt local search to post-process the solutions obtained by DRL-MOA, leading to the results of DRL-MOA+LS. It is noted that the 2-opt is conducted only once for each solution and it only costs several seconds in total. Thus, the results of the experiments on MOGLS-100, MOGLS-200, MOGLS-300, DRL-MOA, and DRL-MOA+LS are presented in Table V. For clarity, only the PFs obtained by MOGLS-100, MOGLS-200, and our method are visualized in Fig. 15.

It is found that using local search to post-process the solutions can further improve the performance. It can be observed that DRL-MOA+LS outperforms the compared MOGLS on all instances while requiring much less computational time. DRL-MOA without local search can also outperform MOGLS on all 200-city instances even the MOGLS has run for 1000 s, while DRL-MOA only requires about 13 s. Although MOGLS performs slightly better than DRL-MOA on 100-city instances, DRL-MOA can always obtain a comparable result within 7 s.

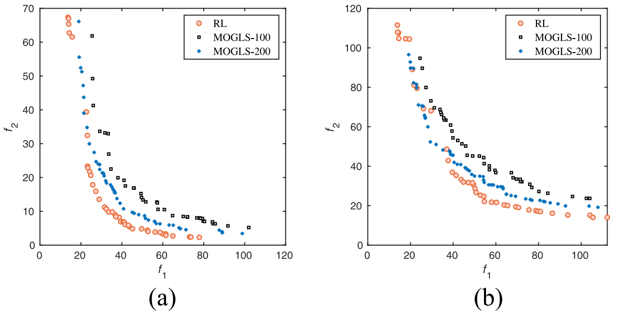


Fig. 15. PFs obtained by MOGLS-100, MOGLS-200, and DRL-MOA on 200-city biobjective TSP instances. (a) PFs of Mixed instances. (b) PFs of Euclidean instances.

The results of the experiments indicate the fast computing speed and guaranteed performance of the DRL-MOA method. Moreover, using local search to post-process the solutions can further improve the performance.

E. Effectiveness of Parameter-Transfer Strategy

In this section, the effectiveness of the neighborhood-based parameter-transfer strategy is checked experimentally. First, the performances of the models that are trained with and without the parameter-transfer strategy are compared. They are both trained on 120 000 20-city MOTSP instances for five epochs. PFs obtained by the two models are presented in Fig. 16(a). It is obvious that the performance of the model is dramatically poor if the parameter-transfer strategy is not used for its training.

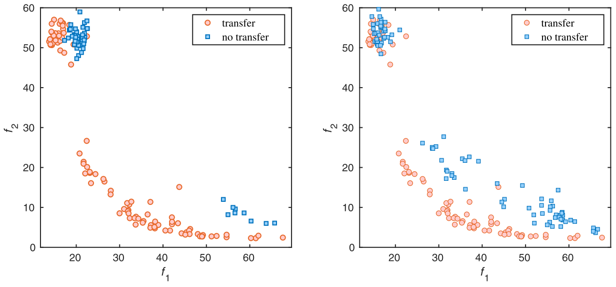
Moreover, we train the model without applying the parameter-transfer strategy on 240 000 instances for ten epochs, that is, the model is trained four times longer than before. The result is presented in Fig. 16(b). It can be seen that, without the parameter-transfer strategy, even if the model is trained four times longer, it still exhibits a poor performance. Thus, it is effective and efficient to apply the parameter-transfer strategy for training; otherwise, it is impossible to obtain a promising model within a reasonable time.

F. Impact of Training on Different Number of Cities

The forgoing models are trained on 40-city instances. In this part, we try to figure out whether there is any difference of the model performance if the model is trained on 20-city instances. The performance of the model that is trained on 20-city instances is presented in Fig. 17(a).

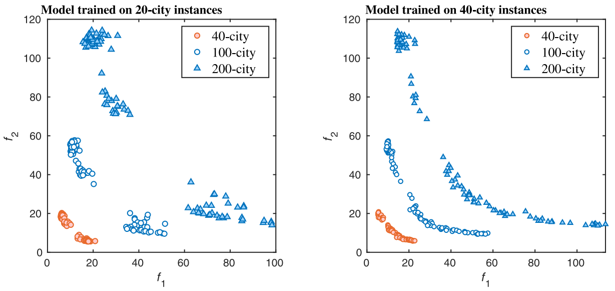
It can be observed that the model trained on 20-city instances exhibits an apparently worse performance than the one trained on 40-city instances. A large number of solutions obtained by the 20-city model are crowded in several regions and there are less nondominated solutions. A possible reason for the deteriorated result is that, when training on 40-city instances, 40-city selecting decisions are made and evaluated in the process of training each instance, which are twice of that when training on 20-city instances. Loosely speaking, if

[4] https://github.com/kevin031060/Genetic_Local_Search_TSP



(a) (b)

Fig. 16. Performances of the models that are trained with and without the parameter-transfer strategy. (a) Performances of two models: one is trained via the parameter-transfer strategy; and another is trained without transferring the network weights. They are both trained on 120 000 instances for five epochs. (b) Performances of two models: the first model is the same with that in (a); and the second model is trained on 240 000 instances for ten epochs without applying the parameter-transfer strategy.



(a) (b)

Fig. 17. Two models trained, respectively, on 20- and 40-city Euclidean biobjective TSP instances. They are used to approximate the PF of 40-, 100-, and 200-city problems. (a) Model trained on 20-city instances. (b) Model trained on 40-city instances.

both the two models use 120 000 instances, the 40-city model is trained based on 120 000 × 40 cities which are twice of that of the 20-city model. Therefore, the model trained on 40-city instances is better. We can simply increase the number of training instances to improve the performance.

Finally, it is also interesting to see that the solutions output by DRL-MOA are not all nondominated. Moreover, these solutions are not distributed evenly (being along with the provided search directions). These issues deserve more studies in the future.

G. Summary of the Results

Observed from the experimental results, we can conclude that the DRL-MOA is able to handle MOTSP both effectively and efficiently. In comparison with classical multiobjective optimization methods, DRL-MOA has shown some encouragingly new characteristics, for example, strong generalization ability, fast solving speed, and promising quality of the solutions, which can be summarized as follows.

1) *Strong Generalization Ability:* Once the trained model is available, it can scale to newly encountered problems with no need of retraining the model. Moreover, its performance is less affected by the increase in the number of cities compared to existing methods.

2) *A Better Balance Between the Solving Speed and the Quality of Solutions:* The Pareto-optimal solutions can be always obtained within a reasonable time while the quality of the solutions is still guaranteed.

V. Conclusion

Multiobjective optimization, appeared in various disciplines, is a fundamental mathematical problem. Evolutionary algorithms have been recognized as suitable methods to handle such problem for a long time. However, evolutionary algorithms, as iteration-based solvers, are difficult to be used for online optimization. Moreover, without the use of a large number of iterations and/or a large population size, evolutionary algorithms can hardly solve large-scale optimization problems [13]–[15].

Inspired by the very recent work of DRL for the single-objective optimization, this article provides a new way of solving the MOP) by means of DRL and has found very encouraging results. In specific, on MOTSP instances, the proposed DRL-MOA significantly outperforms NSGA-II, MOEA/D, and MOGLS in terms of the solution convergence, spread performance, as well as the computing time, and thus, making a strong claim to use the DRL-MOA, a noniterative solver, to deal with MOPs in the future.

With respect to the future studies, first, in the current DRL-MOA, a 1-D convolution layer which corresponds to the city information is used as inputs. Effectively, a distance matrix used as inputs can be further studied, that is, using a 2-D convolution layer. Second, the distribution of the solutions obtained by the DRL-MOA is not as even as expected. Therefore, it is worth investigating how to improve the distribution of the obtained solutions. Overall, multiobjective optimization by DRL is still in its infancy. It is expected that this article will motivate more researchers to investigate this promising direction, developing more advanced methods in the future.

References

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[2] Q. Zhang and H. Li, “MOEA/D: A multiobjective evolutionary algorithm based on decomposition,” *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, Dec. 2007.

[3] L. Ke, Q. Zhang, and R. Battiti, “MOEA/D-ACO: A multiobjective evolutionary algorithm using decomposition and antcolony,” *IEEE Trans. Cybern.*, vol. 43, no. 6, pp. 1845–1859, Dec. 2013.

[4] B. A. Beirigo and A. G. dos Santos, “Application of NSGA-II framework to the travel planning problem using real-world travel data,” in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Vancouver, BC, Canada, 2016, pp. 746–753.

[5] W. Peng, Q. Zhang, and H. Li, “Comparison between MOEA/D and NSGA-II on the multi-objective travelling salesman problem,” in *Multi-Objective Memetic Algorithms*. Heidelberg, Germany: Springer, 2009, pp. 309–324.

[6] S. Lin and B. W. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem,” *Oper. Res.*, vol. 21, no. 2, pp. 498–516, 1973.

[7] D. Johnson, “Local search and the traveling salesman problem,” in *Proc. 17th Int. Colloquium Automata Lang. Program. Lecture Notes Comput. Sci.*, 1990, pp. 443–460.

[8] E. Angel, E. Bampis, and L. Gourvès, "A dynasearch neighborhood for the bicriteria traveling salesman problem," in *Metaheuristics for Multiobjective Optimisation*. Heidelberg, Germany: Springer, 2004, pp. 153–176.

[9] A. Jaszkiewicz, "On the performance of multiple-objective genetic local search on the 0/1 knapsack problem—A comparative experiment," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 402–412, Aug. 2002.

[10] L. Ke, Q. Zhang, and R. Battiti, "Hybridization of decomposition and local search for multiobjective optimization," *IEEE Trans. Cybern.*, vol. 44, no. 10, pp. 1808–1820, Oct. 2014.

[11] X. Cai, Y. Li, Z. Fan, and Q. Zhang, "An external archive guided multiobjective evolutionary algorithm based on decomposition for combinatorial optimization," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 508–523, Aug. 2015.

[12] X. Cai, H. Sun, Q. Zhang, and Y. Huang, "A grid weighted sum Pareto local search for combinatorial multi and many-objective optimization," *IEEE Trans. Cybern.*, vol. 49, no. 9, pp. 3586–3598, Sep. 2018.

[13] T. Lust and J. Teghem, "The multiobjective traveling salesman problem: A survey and a new approach," in *Advances in Multi-Objective Nature Inspired Computing*. Heidelberg, Germany: Springer, 2010, pp. 119–141.

[14] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "A decision variable clustering-based evolutionary algorithm for large-scale many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 97–112, Feb. 2018.

[15] M. Ming, R. Wang, and T. Zhang, "Evolutionary many-constraint optimization: An exploratory analysis," in *Proc. Int. Conf. Evol. Multi Criterion Optim.*, 2019, pp. 165–176.

[16] D. H. Wolpert, W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[17] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9839–9849.

[18] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2692–2700.

[19] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014. [Online]. Available: arXiv:1409.0473.

[20] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016. [Online]. Available: arXiv:1611.09940.

[21] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" 2018. [Online]. Available: arXiv:1803.08475.

[22] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the TSP by policy gradient," in *Proc. Int. Conf. Integr. Constraint Program. Artif. Intell. Oper. Res.*, 2018, pp. 170–181.

[23] C.-H. Hsu *et al.*, "MONAS: Multi-objective neural architecture search using reinforcement learning," 2018. [Online]. Available: arXiv:1806.10332.

[24] H. Mossalam, Y. M. Assael, D. M. Roijers, and S. Whiteson, "Multi-objective deep reinforcement learning," 2016. [Online]. Available: arXiv:1610.02707.

[25] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[26] T. Murata, H. Ishibuchi, and M. Gen, "Specification of genetic search directions in cellular multi-objective genetic algorithms," in *Proc. Int. Conf. Evol. Multi Criterion Optim.*, 2001, pp. 82–95.

[27] K. Li, K. Deb, Q. Zhang, and S. Kwong, "An evolutionary many-objective optimization algorithm based on dominance and decomposition," *IEEE Trans. Evol. Comput.*, vol. 19, no. 5, pp. 694–716, Oct. 2015.

[28] J. Chen, J. Li, and B. Xin, "Dmoea-εC : Decomposition-based multiobjective evolutionary algorithm with the ε-constraint framework," *IEEE Trans. Evol. Comput.*, vol. 21, no. 5, pp. 714–730, Oct. 2017.

[29] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Aug. 2014.

[30] K. Miettinen, *Nonlinear Multiobjective Optimization*, vol. 12. New York, NY, USA: Springer, 2012.

[31] R. Wang, Z. Zhou, H. Ishibuchi, T. Liao, and T. Zhang, "Localized weighted sum method for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 3–18, Feb. 2018.

[32] R. Wang, Q. Zhang, and T. Zhang, "Decomposition-based algorithms using Pareto adaptive scalarizing methods," *IEEE Trans. Evol. Comput.*, vol. 20, no. 6, pp. 821–837, Dec. 2016.

[33] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[34] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," 2014. [Online]. Available: arXiv:1406.1078.

[35] V. R. Konda and J. N. Tsitsiklis, "Actor–Critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[36] Y. Tian, R. Cheng, X. Zhang, and Y. Jin, "PlatEMO: A MATLAB platform for evolutionary multi-objective optimization," *IEEE Comput. Intell. Mag.*, vol. 12, no. 4, pp. 73–87, Nov. 2017.

[37] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, 1991.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: arXiv:1412.6980,

[39] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Stat.*, 2010, pp. 249–256.

[40] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 28, no. 3, pp. 392–403, Aug. 1998.

[41] A. Jaszkiewicz, "Genetic local search for multi-objective combinatorial optimization," *Eur. J. Oper. Res.*, vol. 137, no. 1, pp. 50–71, 2002.

Kaiwen Li received the B.S. and M.S. degrees from the National University of Defense Technology (NUDT), Changsha, China, in 2016 and 2018, respectively.

He is a student with the College of Systems Engineering, NUDT. His research interests include prediction technique, multiobjective optimization, reinforcement learning, data mining, and optimization methods on energy Internet.

Tao Zhang received the B.S., M.S., and Ph.D. degrees from the National University of Defense Technology (NUDT), Changsha, China, in 1998, 2001, and 2004, respectively.

He is a Professor with the College of Systems Engineering, NUDT. His research interests include multicriteria decision making, optimal scheduling, data mining, and optimization methods on energy Internet network.

Rui Wang received the B.S. degree from the National University of Defense Technology (NUDT), Changsha, China, in 2008, and the Ph.D. degree from the University of Sheffield, Sheffield, U.K., in 2013.

He is a Lecturer with the College of Systems Engineering, NUDT. His research interests include evolutionary computation, multiobjective optimization, machine learning, and various applications using evolutionary algorithms.