Solving Orienteering Problems by Hybridizing Evolutionary Algorithm and Deep Reinforcement Learning

Rui Wang, Wei Liu, Kaiwen Li, Tao Zhang, Ling Wang, and Xin Xu

***Abstract*—The orienteering problem (OP) is widely applied in real life. However, as the scale of real-world problem scenarios grows quickly, traditional exact, heuristics and learning-based methods have difficulty balancing optimization accuracy and efficiency. This study proposes a problem decomposition-based double-layer optimization framework named DEA-DYPN to solve OPs. Using a diversity evolutionary algorithm (DEA) as the external optimizer and a dynamic pointer network (DYPN) as the inner optimizer, we significantly reduce the difficulty of solving large-scale OPs. Several targeted optimization operators are innovatively designed for stronger search ability, including a greedy population initialization heuristic, an elite strategy, a population restart mechanism, and a fitness-sharing selection strategy. Moreover, a dynamic embedding mechanism is introduced to DYPN to improve its characteristic learning ability. Extensive comparative experiments on OP instances with sizes from 20 to 500 are conducted for algorithmic performance validation. More experiments and analyses including the significance test, stability analysis, complexity analysis, sensitivity analysis, and ablation experiments are also conducted for comprehensive algorithmic evaluation. Experimental results show that our proposed DEA-DYPN ranks first according to the Friedman test, and outperforms the competitor algorithms at most 69%.**

***Impact Statement*—The orienteering problem (OP) has many real-life applications such as tourism route planning, home fuel delivery, unmanned aerial vehicle reconnaissance mission planning, and facility inspection. However, traditional exact algorithms, heuristics, and learning methods have difficulty balancing optimization efficiency and accuracy, especially for large-scale cases. This study proposes a problem decomposition-based double-layer optimization framework named DEA-DYPN to solve OPs. By combining the evolutionary algorithm and deep reinforcement learning method, we significantly reduce the difficulty of solving large-scale OPs. DEA-DYPN significantly outperforms the compared representative algorithms in experiments, especially in instances with more than 50 nodes. This hybrid optimization approach provides a new idea for solving traditional combinatorial optimization problems. The modularity design facilitates its expansion to other variants of the OP.**

***Index Terms*—Orienteering Problem, Decomposition, Evolutionary Algorithm, Deep Reinforcement Learning, Attention.**

The authors gratefully acknowledge the financial support provided by the Major Open Project of Xiangjang Laboratory (No. 22XJ02003), the National Natural Science Foundation of China (No. 62122093), the Science and Technology Project for Young and Middle-aged Talents of Hunan (2023TJ-Z03), the Science and Technology Innovation Program of Hunan Province (No. 2023RC1002), the Scientific Project of NUDT (ZK21-07, 23-ZZCX-JDZ-28), and the National Natural Science Foundation of China (No. 72071205). *(Corresponding authors: Wei Liu.)*

Rui Wang is with the College of System Engineering, the National University of Defense Technology, Changsha 410073, P.R. China, and with the Xiangjiang Laboratory, Changsha 410205.

Wei Liu, Kaiwen Li, and Tao Zhang are with the College of System Engineering, the National University of Defense Technology, Changsha 410073, P.R. China, and with the Hunan Key Laboratory of Multi-Energy System Intelligent Interconnection Technology, HKL-MSI2T, Changsha 410073, P.R. China (e-mail: rui_wang@nudt.edu.cn, weiliu16@nudt.edu.cn, and kaiwenli_nudt@foxmail.com); Ling Wang is with the Department of Automation, Tsinghua University, Beijing, 100084, P.R. China (wangling@mail.tsinghua.edu.cn); Xin Xu is with the College of Intelligence Science and Technology, National University of Defense Technology, Changsha 410073, P.R. China (xinxu@nudt.edu.cn).

I. Introduction

As one of the most classical combinatorial optimization problems (COPs), the orienteering problem (OP) has arisen widely in recent years. Many real-world problems are modeled as OP problems to be solved, including the tourism route planning [1], home fuel delivery [2], unmanned aerial vehicle reconnaissance mission planning [3], facility inspection[4], and so forth. Defining an OP by a directed graph $G = (V, E)$, where V represents the set of nodes and E represents the set of edges, the objective is to find a path $P = (v_1, v_2, \ldots, v_k)$ that satisfies the following criteria:

- The total path length is within the given limit;
- The path starts and ends at a fixed node, which is called a depot;
- The path visits each node at most once;
- The path maximizes the total score collected.

The optimization model for the OP can be formulated as:

$$\text{Maximize} \quad f(v_1, v_2, \ldots, v_k) = \sum_{i=1}^{k} s_{v_i}, \tag{1}$$

subject to:

$$\sum_{i=1}^{k-1} t_{v_i v_{i+1}} \leq T, \tag{2}$$

$$v_1 = v_k = Depot, \tag{3}$$

$$\{v_i | i = 2, 3, \ldots, k-1\} \subseteq \{1, 2, \ldots, N\}, \tag{4}$$

where s_{v_i} is the score associated with the node v_i that can be collected, $t_{v_i v_{i+1}}$ is the distance from the node v_i to the node v_{i+1}, T is the maximum tour length for the exploration, and $Depot$ is the node where the exploration starts and ends.

Whereas the OP has been studied for decades, existing methods still have difficulty balancing efficiency and accuracy, due to the nondeterministic polynomial NP-hard property of the OP. Existing exact, heuristic, and learning-based methods are the three most used approaches. Exact methods, while

guaranteeing optimal solutions, suffer from exponential growth in computational complexity with the increase in problem size, rendering them impractical for large instances. Heuristic methods, known for their efficiency in generating satisfactory solutions, often struggle to maintain solution quality as the problem scale increases. Learning-based approaches, despite their promise in leveraging historical data and patterns, face challenges in generalizing across diverse and complex OP scenarios, often requiring extensive training data and computational resources. These traditional approaches face a common challenge: the difficulty in striking a balance between exploration (i.e., searching new areas of the solution space) and exploitation (i.e., refining known good solutions), particularly in vast solution landscapes of large-scale OPs. This balancing act is crucial for avoiding local optima and ensuring that the solution space is thoroughly explored for potential global optima. However, the inherent characteristics of the OP, combined with its NP-hard nature, exacerbate this challenge, leading to low-quality solutions or excessive computational demands.

Reinforcement learning (RL) methods exhibit significant advantages in solving complex optimization problems due to their ability to dynamically adapt to varying problem landscapes through iterative learning and feedback mechanisms. Unlike traditional optimization techniques, RL continuously improves its strategies based on rewards obtained from previous iterations, enabling effective exploration-exploitation trade-offs. This adaptability leads to enhanced convergence rates and superior solution quality in complex, high-dimensional search spaces. Integrating RL with meta-heuristic methods further amplifies these benefits by leveraging the global search capabilities of meta-heuristics. This hybrid approach not only improves robustness and efficiency in dynamic and stochastic environments but also provides a powerful framework for tackling a broader range of real-world optimization challenges. Numerous studies have demonstrated the efficacy of RL in improving meta-heuristic algorithms [5], [6].

This study proposes to solve the OP through a decomposition strategy, i.e., solving simpler subproblems collaboratively with targeted approaches. Since an OP is essentially a combination of node selection and path planning [2], it can be decomposed into a knapsack problem (KP) and a traveling salesman problem (TSP). A double-layer optimization framework is proposed to solve them. We design a diversity evolutionary algorithm (DEA) as the external optimizer for solving the KP, and a dynamic pointer network (DYPN) as the internal optimizer for solving the TSP. DEA prevents the algorithm from falling into local optima by adding a population restart mechanism, a fitness-sharing selection strategy to the traditional genetic algorithm (GA). Meanwhile, a greedy heuristic is used to initialize a high-quality population, and an elite strategy is used to ensure that dominant solutions are not ruled out during evolution. DYPN improves the traditional pointer network (PN) by introducing dynamic information in the planning process, effectively accelerating model learning. A parallel matrix-based computation and GPU acceleration approach is used to accelerate the computation. While DEA selects nodes to maximize the score obtained, DYPN plans paths and helps the former judge feasibility and make further adjustments. Iterative update generates the final solution. The proposed framework is called DEA-DYPN.

Such an algorithmic design greatly simplifies the problem-solving difficulty. Since DYPN can plan TSP paths for a whole population immediately and parallelly just through a network feed-forward, the OP can be reduced to the KP, which is much simpler for DEA to solve. The global search capability of DEA can therefore be better utilized. Extensive comparison experiments, significance tests, stability analysis, complexity analysis, sensitivity analysis, and ablation experiments are conducted to verify the scientific validity and effectiveness of the proposed methodology. Experimental results show that our proposed DEA-DYPN ranks first according to the Friedman test, and outperforms the competitor algorithms at most 69%. The main contributions of our work can be summarised as follows:

- A double-layer optimization framework based on problem decomposition is proposed for solving OPs. By decomposing the OP into a KP and a TSP and solving them interactively and iteratively, the proposed DEA-DYPN can deal effectively with large-scale OPs.
- A diversity evolutionary algorithm is proposed to maintain high population diversity during evolution. Several targeted operators and mechanisms are proposed to enhance population diversity and accelerate convergence.
- A dynamic pointer network is proposed to accelerate the model learning capacity. The result of the stepped decision is taken into account dynamically in the form of a distance during decoding, guiding the agent to implement a better action instead of exploring completely aimlessly.
- An algorithmic acceleration design based on matrix computing and parallel computing is proposed for the interaction between DEA and DYPN.
- Extensive computational experiments, significance tests, stability analysis, complexity analysis, sensitivity analysis, and ablation experiments are conducted to validate the strong learning ability, optimization ability, and generalization ability of the proposed DEA-DYPN.

The rest of this study is structured as follows: Section II reviews the related work and points out the necessity of our work. Section III elaborates the proposed DEA-DYPN framework, including the DEA module, the DYPN module, and the REINFORCE training algorithm. The experimental setup is described in Section IV, and the experimental results are analyzed in Section V. Model performance, complexity analysis, learning ability analysis, sensitivity analysis, and ablation experiments are included. Section VI concludes this study and identifies several future directions.

II. Related Work

A. Traditional Methods for Orienteering Problems

While the OP has been studied for decades, exact and heuristic algorithms are most commonly used. Exact algorithms, such as the branch-and-bound [7], [8], cutting plane [9]

and branch-and-cut algorithms [10], [11], can obtain optimal solutions for simple cases, however, have a high demand in terms of the problem structure. Nonlinearities in the problem structure and discontinuities in the decision variables can greatly increase the computational difficulty of exact algorithms. Moreover, growth in the problem scale will cause the search space to explode. This makes exact algorithms weak for large-scale cases.

In contrast, heuristics avoid rigorous mathematical modeling of the problem and search for solutions iteratively. The computational resources they require are generally linearly related to the problem size. Since they do not search for optimal but satisfactory solutions, heuristics can often obtain solutions quickly. The evolutionary algorithm (EA), particle swarm optimization (PSO) algorithm, neighborhood search (NS) algorithm, local search (LS) algorithm, Tabu search (TS) algorithm, and their variants are the most popular heuristics in recent years. Gunawan et al. [12] improved the traditional iterative local search (ILS) [13] for the OP by introducing several local research options, which were demonstrated effective in improving the algorithm performance. Kobeaga et al. [14] proposed an EA with infeasible individuals for solving the OP. The well-designed coding mode, operators, and Lin-Kernighan heuristic made it work well. Pvenivcka et al. [15] considered three variations of the variable neighborhood search (VNS) method. The reduced VNS showed the best performance on their OP benchmark instances. Chou et al. [16] proposed to embed a Monte Carlo evaluator into a Tabu search algorithm and got satisfactory effectiveness. Facing the team orienteering problem with time windows, Amarouche et al. [17] improved the traditional large neighborhood search (LNS) method by maintaining two different search spaces and using a long-term memory mechanism to keep high-quality routes encountered in elite solutions.

Nevertheless, most of these works focus on small-scale OP optimization and usually take a long-term iterative optimization process. This means these algorithms usually apply in scenarios with sufficient decision time. Moreover, their optimization operators are often designed for specific instances and have weak generalization ability. Once the data is changed, a significant performance decrease may occur.

B. Deep Reinforcement Learning for Combinatorial Optimization

Deep reinforcement learning (DRL) has been demonstrated effective for COPs. Vinyals et al. [18] first introduced a PN to solve a TSP via supervised learning. Bello et al. [19] proposed to train neural networks through an RL method. By learning parameters from trial and error, RL requires no labels and performs well on the TSP and the KP. Nazari et al. [20] introduced an attention mechanism into the PN and made the model suitable for the vehicle routing problem (VRP). Li et al.[21] further extended the PN to multi-objective TSPs by introducing a problem-decomposition strategy and a parameter migration method. Kool et al. [22] proposed an attention model (AM) and trained it by REINFORCE with a greedy rollout baseline to solve some common path planning problems. Once the model is trained, the real-time generation of solutions is a major advantage of this type of end-to-end learning approach. Xu et al. [23] proposed a reinforcement learning method with multiple relational attention for solving several classical vehicle routing problems. Batch normalization reordering, gate aggregation, and dynamic-aware context embedding were applied to improve the learning ability.

While several works have applied DRL methods on OPs [23], [22], the studied cases are on a small scale (i.e., no more than 100 nodes), and exhibit mediocre optimization performance. Complex problem structures and large-scale problem scenarios place high demands on algorithm design.

C. Hybridizing Methods

Compared with a single DRL or heuristic method, hybridizing methods show more potential in recent studies. Machine learning methods such as DRL excel in improving search efficiency, selecting optimization operators, and improving solution quality. Various combinations of heuristics and DRL methods show good performance in different problems.

Some studies used DRL models to optimize the selection of operators or controlling parameters in heuristics for higher generalization ability: Lu et al. [24] proposed a "Learn to Improve" (L2I) framework for the VRP. RL in that work was used as a controller to select optimization operators. L2I showed higher effectiveness than classical operations research (OR) algorithms according to the experimental results. Similarly, Qi et al. [25] used a Q-leaning algorithm to select local search operators in the evolutionary process while solving the VRP with time windows. Zhang et al. [26] proposed a heuristic EA based on deep Q-network (DQN) to solve the energy-aware multi-AGV flexible job shop scheduling problem, where DQN learns the experience of selecting local search operators. Moreover, RL has also been used for parameter self-adjustment within non-dominated sorting genetic algorithm-II (NSGA-II) for multi-objective optimization problems (MOPs) [27], [28].

Some studies used DRL models to learn new heuristics for stronger characteristic learning ability: Kallestad et al. [29] used a DRL model as an alternative to the adaptive layer of adaptive large neighborhood search. The DRL agent in that work can consider additional information from the search process to make better decisions. Wu et al. [30] proposed a DRL model to learn improved heuristics for routing problems. Such a design outperformed the state-of-the-art deep learning-based approaches. Liang et al. [31] combined DRL and EA to solve MOPs with irregular Pareto Fronts. DRL was used to adaptively adjust the distribution of reference vectors.

Moreover, some studies decomposed a specific problem, and used heuristics and DRL methods respectively to solve the subproblems: Watkins et al. [32] used deep Q-learning together with a graph neural network for feature extraction, helping heuristics solve the graph coloring problem. Liu et al. [33] proposed an algorithmic combination of the DRL and NSGA-II for solving the multi-objective OP (MO-OP). NSGA-II updated the choice of nodes through the dominance relation of individuals. More discussion about RL-assisted evolutionary algorithms can be found in [34], [35].

These algorithms perform well on their studied problems and provide us with good ideas for algorithm design. However, they are not exactly suitable for the large-scale single-objective OP, which has higher demands on algorithmic search efficiency and diversity maintenance ability. Therefore, our study aims to propose a new method that can balance the effectiveness and efficiency of the algorithm, and thus can effectively solve large-scale OP cases with strong generalization ability. Considering the natural structure of the OP, we agree that the problem decomposition-based optimization idea is efficient for dealing with large-scale cases, as stated in the literature [33]. However, different from solving an MO-OP based on dominance relations, we need to pay much more attention to population diversity when facing large-scale single-objective OP. Moreover, the interaction efficiency between optimizers (i.e., heuristics and DRL modules) is also important. The differences between our work and traditional ones can be summarised as follows:

- A double-layer optimization framework (i.e., DEA-DYPN) specific for the single-objective OP based on problem decomposition.
- An enhanced evolutionary algorithm (i.e., DEA) with multiple diversity operators to enhance the population diversity and prevent the population from falling into a local optimum.
- An enhanced deep neural network (i.e., DYPN) for stronger characteristic learning ability.
- A parallel matrix-based computation and GPU acceleration approach for much more efficient operation.

III. Proposed Model

Algorithm 1 illustrates the pseudo-code of the proposed DEA-DYPN. The model pre-training and population initialization are required in advance. During the evolution, node permutations are first generated by the pre-trained DYPN model based on the initially selected nodes. After calculating fitness, DEA then adjusts its population. After updating solutions for a specified number of generations, the final solution of the OP can be obtained.

Algorithm 1 General Framework of DEA-DYPN

Input: instance set $\mathcal{M}$, maximum number of generations $MaxGen$, population size N

```
Initialize gen ← 1
Train the DRL model for the TSP: DYPN ← REINFORCE(M)
Pop ← Initialization(N)
while gen ≤ MaxGen do
    Route, Length ← DYPN(Pop)
    Pop ← DEA(Pop, Route, Length, N)
    gen ← gen + 1
end while
Route, Length ← DYPN(Pop)
Arc ← UpdateArc(Pop, Route, Length)
```

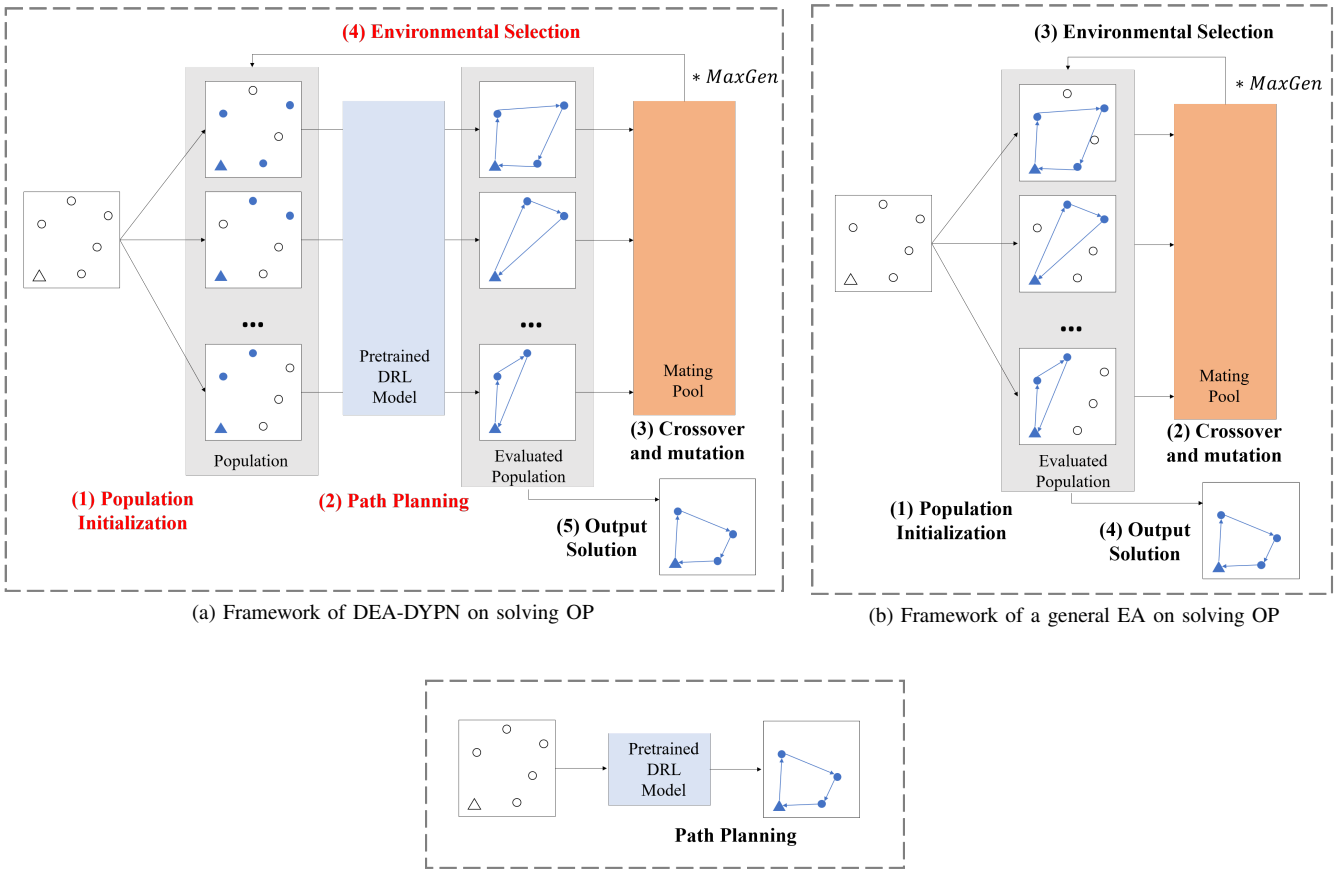
Fig. 1 describes in detail the flowcharts of the DEA-DYPN, a general EA, and a general DRL method in solving an OP. These flowcharts clearly show the elements, as well as their interaction, of these competitor algorithms. Unlike the single-level iterative optimization method EA or end-to-end DRL, the proposed DEA-DYPN is a double-layer iterative optimization framework. Such a problem decomposition-based optimization framework means that each sub-optimizer is responsible for solving a much simpler subproblem. Intuitively, DEA-DYPN seems to have a higher algorithmic complexity. However, a more efficient search would mean much fewer iterations required. A detailed algorithm complexity analysis will be stated in Section V-B.

One of the most significant advantages of this framework is its modularity and simplicity for users. In principle, any heuristic can be used as the external optimizer, and other DRL-based methods can be used as the inner optimizer. In addition, this framework is also suitable for all kinds of variants of the OP by simply adding appropriate constraints to the external optimizer or the inner optimizer. Next, we elaborate on the details of DEA and DYPN.

A. Diversity Evolutionary Algorithm

Evolutionary algorithms have shown good performance in dealing with combinatorial optimization. However, the optimization operators in EAs are generally required to be carefully designed as the problem size increases. Common challenges faced by EAs on large-scale problems include the slow speed of iterative optimization, and the tendency of falling into local optima [36]. Therefore, for Knapsack problems with hundreds of nodes, this study improves a conventional GA by introducing several effective search operators, including a greedy population initialization heuristic, an elite strategy, a population restart mechanism, and a fitness-sharing selection strategy. The greedy population initialization heuristic is used to generate high-quality and high-diversity initial solutions. The elite strategy is used to retain optimal individuals during the evolution. The last two are designed to increase the population diversity, helping the algorithm jump out of local optima. The framework of DEA is described in Algorithm 2.

A binary coding mode is used here to code chromosomes. Among the employed operators, the stochastic universal sampling method is used as the $MateSelection$ operator, which randomly samples N individuals from the current population into the mating pool. Other operators such as the roulette wheel selection, the tournament selection, and the rank-based selection are also available. Their difference can be found in [37]. The uniform distribution crossover operator (Xovud) is used as the $Crossover$ operator with a crossover probability of P_c, and the binary chromosomes mutation operator (Mutbin) is used as the $Mutation$ operator with a mutation probability of P_m. After combining the original population and the generated offspring, fitness values, and the shared fitness values are calculated in order. The population of the next generation is selected according to the shared fitness values. The N individuals with the highest shared fitness values will be selected for the next generation. The population needs to be restarted when the optimal fitness value stagnates for the given



(a) Framework of DEA-DYPN on solving OP

(b) Framework of a general EA on solving OP

(c) Framework of a general DRL on solving OP

Fig. 1. Schematic flowcharts for solving an OP by DEA-DYPN, a general EA, and a general DRL method. (a) DEA-DYPN: five steps are included: population initialization, path planning, crossover and mutation, environmental selection, and output solution. While the DEA module updates node selection iteratively, the pre-trained DYPN model plans paths and calculates their fitness concurrently. The newly proposed modules are labeled in red, including the population initialization module, the path planning module, and the environmental selection module. The detailed operators will be illustrated later; (b) EA: four steps are included: population initialization, crossover and mutation, environmental selection, and output solution. Each individual represents a path; (c) DRL: taking the node information as input, the pre-trained DRL can directly output a feasible path.

generations. Other good environmental selection strategies can be found in [38], [39], [40]. Finally, the $UpdateArc$ operator selects the optimal feasible individual Arc from the whole population. The proposed optimization operators, including the greedy population initialization method, the population restart mechanism, and the fitness-sharing selection strategy are described below.

1) Greedy Population Initialization: A greedy population initialization strategy is adopted to speed up the evolution. For each individual, nodes are selected step by step until the individual violates the constraints. In each step, the node is sampled according to one probability vector P^i, where i is the last sampled node. We first define a score density vector $I^i = (I^i_j, j = 1, \ldots, N)$, which is formulated as (5).

$$I^i_j = s_j / t_{ij}, \quad j \in U, \tag{5}$$

where t_{ij} is the Euclidean distance between the node i and j, $s_j$is the score of the node j, and U is the set of unvisited nodes at that time. Since nodes cannot be visited repeatedly, those visited nodes must be excluded. I^i_j indicates how worthy node j is to be selected next when the last node selected is i. The higher the value of I^i_j is, the more likely the node j is to be selected in that step. Normalization of the score density vector generates the probability vector P^i, as formulated in (6).

$$P^i = \mathrm{softmax}(I^i). \tag{6}$$

To ensure the diversity of the initial population, half of the population is generated greedily as mentioned before and the other half is generated randomly. Such a hybrid strategy is empirically demonstrated to be more effective.

2) Population Restart Mechanism: To handle the issue of being trapped in local optima or solution stagnation, the designed population restart strategy is responsible for detecting a lack of progress in fitness over generations and introducing new individuals. Specifically, when the optimal fitness of the population does not change for several generations, this strategy initiates a population restart by preserving elite individuals, randomly generating new individuals for diversity, and reevaluating fitness. Such an operation can inject new genetic material into the population, rejuvenate the search process, and promote exploration. This calculation costs less but performs well in revitalizing the search process, offering a valuable approach to tackling stagnation in genetic algorithms.

Algorithm 2 General Framework of DEA

Input: Maximum number of generations $MaxGen$, population size N, maximum number of stagnation generations $MaxStag$

```
Initialize gen ← 1
Pop ← Initialization(N)
while gen ≤ MaxGen do
    MatingPool ← MateSelection(Pop)
    Off ← Crossover(MatingPool)
    Off ← Mutation(Off)
    Pop ← Combine(Pop, Off)
    Fit ← FitnessCal(Pop)
    Fit ← FitnessShare(Pop, Fit)
    Pop ← PopSelection(Pop, Fit, N)
    if Fit stagnates for MaxStag generations then
        Pop ← Restart(Pop)
    end if
    gen ← gen + 1
end while
Arc ← UpdateArc(Pop)
```

In this strategy, the allowed maximum number of stagnation generations and the proportion of elite individuals retained are two hyper-parameters that must be set artificially.

3) Fitness-sharing Selection Strategy: To enhance the exploration capability of an EA, the proposed fitness-sharing mechanism employs a shared fitness function to promote diversity by penalizing individuals who are too close to each other in the search space. The shared fitness values are computed by calculating the Euclidean distances among individuals and applying a sharing function.

Given a population with original fitness values, denoted as $Fit = (fit_1, fit_2, ..., fit_n)$, and the distance between any individual i and individual j, denoted as t_{ij}, the shared fitness for individual i is defined as:

$$\bar{fit}_i = \frac{fit_i}{\sum_{j=1}^{n} \max(0, 1 - t_{ij})}. \tag{7}$$

During the environment selection stage, individuals of the next generation are selected according to their shared fitness values. Such a strategy works by penalizing similar individuals to prioritize the selection of individuals that exhibit higher diversity. By combining fitness sharing and diversity-based selection, this strategy effectively encourages the exploration of the search space. Mitigating premature convergence and promoting the preservation of genetic diversity eventually facilitates the search for high-quality solutions. It is worth noting that, while there have already been some good environmental selection strategies such as the fitness-distance balance (FDB) method [38], natural survivor method (NSM) [39], and fitness-distance-constraint (FDC) method [40], our method does not require extra weight factors, and can adaptively balance the optimization quality and diversity.

B. Dynamic Pointer Network

To generate a Hamiltonian path based on the selected nodes, the PN [20] is improved by embedding dynamic information. The DYPN model is formally described as follows. Taking a TSP instance r with N nodes as an example, the solution $\pi = (\pi_1, \ldots, \pi_N)$ is a permutation of all the nodes, the policy $p(\pi|r)$ for generating a solution π can be described below, following the probability chain rule:

$$p_\theta(\boldsymbol{\pi} \mid r) = \prod_{t=1}^{N} p_\theta\left(\pi_t \mid r, \boldsymbol{\pi}_{0\sim t-1}\right), \tag{8}$$

where θ is the model parameter set. DYPN follows an encoder-decoder architecture, where the encoder is used to characterize the embeddings, and the decoder is used to generate a node permutation. It is worth mentioning that the position of the depot in the permutation is less important as the path can always form a loop.

1) Encoder: $1D$ convolution layers are used as the encoder to map the inputs into D-dimensional vectors ($D = 128$ in this work). In each $1D$ convolution layer, the number of in-channels is the input length, the number of filters is set as D, which is also the number of out-channels, and there is one kernel for each filter. It is worth mentioning that multiple $1D$ convolution layers are utilized for different input elements and they are shared among these inputs.

In this study, we embed not only static features (i.e., node locations) but also dynamic features, which are defined as the distances between the last visited node and other nodes. Defining the input set as $X = (x^i, i = 1, \ldots, N)$, where x^i is a sequence of tuples $(x_t^i = \left(s^i, d_t^i\right), t = 0, \ldots, T)$. s^i and d_t^i are the static and dynamic features, respectively.

Static embedding. Since there are two elements in each node location, i.e., x-coordinate and y-coordinate, the input length of each static embedding is 2. For each node i, its location vector s^i is mapped into a D-dimensional vector $\bar{s}^i$. For all N nodes, the static embeddings are set as $\bar{S} = (\bar{s}^i, i = 1, \ldots, N)$.

Dynamic embedding. Defining the dynamic feature for the node i at t time as d_t^i, it represents the distance between the node i and the node decoded in step t. After normalizing the dynamic vector $(d_t^i, i = 1, 2, \ldots, N)$ using the max-min normalization, a $1D$ convolution layer is then applied to map it into a dynamic embedding $(\bar{d_t^i}, i = 1, 2, \ldots, N)$. Since d_t^i is a value, the number of in-channels in the dynamic embeddings is 1.

2) Decoder: The decoder is composed of two parts, i.e., an RNN and a context-based attention module (including an attention layer, a context embedding layer, a log-probability layer, and an inference layer). The RNN is used to record the decoding outputs, and the attention module is used to calculate the node selection probability. Fig. 2 describes the decoder structure as well as the decoding process.

RNN. The gated recurrent unit (GRU) [41], a typical variant of RNN, is used here to record the decoding information. The GRU takes the static embedding $\bar{s}^{\pi_{t-1}}$ of the last decoded node π_{t-1} and its last memory state h_{t-1} (if $t > 1$) as inputs, and outputs a new memory state h_t. This calculation is formulated as (9). Since no node is selected in step $t = 1$, $\bar{s}^{\pi_0}$ is initialized by the embedding of a 2-dimensional zero vector.

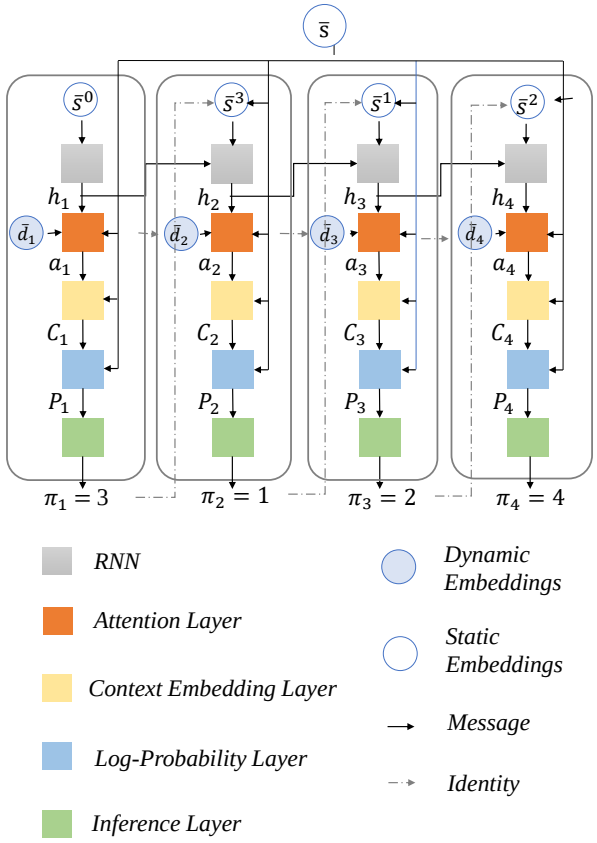


Fig. 2. Illustration of the decoding. The decoder takes both the static and dynamic embeddings as inputs. Dynamic embeddings are only used in the attention layer. This example shows how a tour $\pi = (3, 1, 2, 4)$ is constructed.

$$h_t = f_{GRU}\left(\bar{s}^{\pi_{t-1}}, h_{t-1}\right). \tag{9}$$

Attention layer. An attention layer is proposed to measure how relevant each node might be in each decoding step. Taking the memory state h_t, node static embeddings $\bar{S}$, and the dynamic embedding $\bar{d}_t$ as inputs, the attention layer outputs an alignment vector a_t, which is calculated as (10).

$$a_t = \text{softmax}\left(u_t\right), \tag{10}$$

where

$$u_t = v_a^T \tanh\left(W_a\left[\bar{S}; \bar{d}_t; h_t\right]\right). \tag{11}$$

v_a and W_a are learnable parameters. $\bar{d}_1$ is set as the dynamic embedding of a 1-dimensional zero vector.

Context embedding layer. This layer is used to calculate a context vector c_t, which represents the weighted static embeddings. The inputs of the context embedding layer are a_t and $\bar{S}$, and the context vector c_t is defined as follows:

$$c_t = a_t \bar{S}. \tag{12}$$

Log-probability layer. Taking the combination of the static embeddings $\bar{S}$ and the context c_t as inputs, the log probability mentioned in (8) for each node can be calculated as follows. v_c and W_c are learnable parameters.

$$p_\theta\left(\pi_t \mid r, \boldsymbol{\pi}_{0\sim t-1}\right) = \text{softmax}\left(\tilde{u}_t\right), \tag{13}$$

where

$$\tilde{u}_t = v_c^T \tanh\left(W_c\left[\bar{S}; c_t\right]\right). \tag{14}$$

Inference layer. The inference layer is used to select nodes based on the log probabilities $p_\theta\left(\pi_t \mid r, \boldsymbol{\pi}_{0\sim t-1}\right)$. Sampling search and greedy search strategies [42] are available. Specifically, the sampling search strategy is used in the model training stage to increase the exploration ability of the model and the greedy search strategy is used in validation to obtain better solutions. In each decoding step, while the sampling search strategy samples one node according to the probability, the greedy search always selects the node with the highest probability.

C. REINFORCE Algorithm

For end-to-end neural network training, there are multiple parameter learning approaches, commonly including supervised learning, unsupervised learning, reinforcement learning, and evolutionary learning. However, considering that supervised learning is expensive to acquire labels [19], unsupervised learning does not align directly with specific task objectives [43], and evolutionary algorithms are difficult to train deep neural networks with such large-scale parameters [33], these methods are not suitable for training the proposed DYPN. The RL, however, offers a robust alternative by allowing the network to learn optimal policies through interaction with the environment, thereby directly optimizing the desired performance metrics without the need for extensive labeled data.

The REINFORCE algorithm [44], as a classic policy gradient RL approach, is used here to train the DYPN. An actor network and a critic network are included. For a problem instance r, while the actor provides a node permutation $\boldsymbol{\pi}$ and the corresponding reward $L(\boldsymbol{\pi})$, the critic provides an estimated reward $V(k; \phi)$, where ϕ is the parameter set of the critic. In this study, the actor is the DYPN model, and the critic is formed by two dense layers (with a ReLU activation function) and a single output linear layer. The inputs of the critic are static and dynamic embeddings. The training loss is defined as (15) and its gradient is formed as (16).

$$\mathcal{L}(\theta \mid r) = E_{p_\theta(\boldsymbol{\pi}|r)}[L(\boldsymbol{\pi}) - V(r; \phi)], \tag{15}$$

$$\nabla\mathcal{L}(\theta \mid r) = E_{p_\theta(\boldsymbol{\pi}|r)}\left[(L(\boldsymbol{\pi}) - V(r; \phi))\nabla \log p_\theta(\boldsymbol{\pi} \mid r)\right]. \tag{16}$$

Algorithm 3 summarizes the training process. The actor and critic networks are initialized with random parameters θ and ϕ, respectively. M TSP instances are randomly generated as training instances. For each instance k_m, a solution $\boldsymbol{\pi}$ is obtained by the actor. The reward $L^m(\boldsymbol{\pi})$ is defined as the total tour length. The corresponding estimated reward $V(k_m; \phi)$ is calculated by the critic. After calculating the rewards and estimated rewards for all M instances, the parameters of both the actor and the critic are updated. Such parameter updates last for N_{epoch} epochs.

Algorithm 3 Training DYPN by REINFORCE

Input: training size M, number of training epochs N_{epoch}

Initialize the actor network with random weights θ and the critic network with random weights ϕ
Randomly generate M instances
for $epoch = 1 : N_{epoch}$ **do**
  Reset gradients: $d\theta \leftarrow 0, d\phi \leftarrow 0$
  for instance $r_m = r_1, \ldots, r_M$ **do**
    for step $t = 1 : N_{step}$ **do**
      $\boldsymbol{\pi}_t \leftarrow p_\theta(r_m, \boldsymbol{\pi}_{0 \sim t-1})$
    end for
    Calculate reward $L^m(\boldsymbol{\pi})$
    Calculate estimated reward $V(r_m; \phi)$
  end for
  $d\theta \leftarrow \frac{1}{M} \sum_{m=1}^{M} \left(L^m(\boldsymbol{\pi}) - V(r_m; \phi)\right) \nabla_\theta \log p_\theta(\boldsymbol{\pi} \mid r_m)$
  $d\phi \leftarrow \frac{1}{M} \sum_{m=1}^{M} \nabla_\phi \left(L^m(\boldsymbol{\pi}) - V(r_m; \phi)\right)^2$
  Update θ and ϕ respectively using $d\theta$ and $d\phi$
end for

IV. Experimental Setup

All experiments are conducted on a PC with an AMD 8-Core R7-5800H CPU @3.2 GHz and a single RTX 3060 GPU. All code is written in Python 3.8. The proposed DEA is implemented based on the Geatpy [1], which is an open-source library of Python. The source code of this study has been open source [2]. This section introduces the setup of hyper-parameters and competitor algorithms.

A. Hyper-parameters

The learning-based DYPN is trained on 1,280,000 randomly generated 50-node TSP instances for 10 epochs. Setting the batch size as 256 and using Adam [45] as the optimizer with a learning rate of 0.005 and a dropout rate of 0.05, the total training time is about 90 minutes. For each instance, nodes are randomly located within a unit square $[0, 1]*[0, 1]$. The random seed is set as 1234. These hyper-parameters mostly follow the recommended settings of [22]. The learning rate and dropout rate are set according to computational experiments, which will be presented later.

During the testing stage, OP instances with 20, 50, 100, 200, and 500 nodes (except for the depot) are in consideration. Their maximum tour lengths are set as 2, 3, 4, 5, and 6, respectively. Such a setting makes approximately half of the nodes being visited in the optimal solution, which is the most difficult situation for solving [2]. Nodes in test instances are also randomly located within a unit square $[0, 1] * [0, 1]$. The node scores obey a uniform distribution in the range of (0,1) (The depot node has no score). The random seed is set as 12345 for the test instances.

As for the parameters for DEA, the population size is 256 and the maximum number of generations is 400. The probabilities of crossover and mutation are set as 0.9 and 0.01, respectively. Moreover, the maximum number of stagnation generations is set as 20 and the proportion of elite individuals retained is set as 0.5. The maximum running time is set as 300 seconds, which is the same for all iterative competitor algorithms, to ensure fairness in comparison. These hyper-parameters are set according to expert experience and experimental attempts. All the mentioned hyper-parameters are summarized in TABLE I.

TABLE I
Hyperparameter configurations

DYPN		DEA	
Hyperparameter	**Value**	**Hyperparameter**	**Value**
No. of epochs	10	Population size	256
No. of instances	1,280,000	Max No. of generations	400
Batch size	256	Probability of crossover	0.9
Optimizer	Adam	Probability of mutation	0.01
Learning rate	0.005	Max No. of stagnation	20
Dropout rate	0.05	Proportion of restart	0.5
		Max running time (s)	300

B. Competitor Algorithms

To verify the effectiveness of the proposed method, this study implements several classical or state-of-the-art algorithms for comparison, including the Gurobi [46], strengthen elitist genetic algorithm (SEGA), fitness-distance balance-based evolutionary algorithm (FDB-EA) [47], AM [22], and Tsili-Greedy algorithm [48]. These algorithms are the most representative ones respectively for commercial solvers, genetic algorithms, deep reinforcement learning algorithms, and greedy heuristics for solving the OP. A detailed introduction to these competitors is below.

1) Gurobi: Gurobi is a leading commercial optimization solver and is widely recognized for its good performance. In this study, we develop a corresponding mathematical planning model for the OP and run Gurobi for 30 and 300 seconds for the performance comparison. Since Gurobi always tries to find an optimal solution, setting a maximum runtime is necessary.

2) SEGA: SEGA, a classical variant of genetic algorithm, has been demonstrated effective in searching for high-quality solutions in the context of several traditional COPs. It emphasizes the preservation of elite individuals from one generation to the next, thus enhancing the exploitation of promising solutions. This study sets its population size as 256, which is the same as the DEA, and sets the maximum number of generations as 40,000, making its running time not less than the DEA. All the other parameter settings follow the default of the Geatpy.

3) FDB-EA: As one of the latest proposed EAs for route planning, FDB-EA [47] strategically amalgamates three distinct guide selection methods (i.e., greedy, random, and FDB-score-based methods), to dynamically adapt its search behavior across different optimization phases. This hybridization facilitates a balanced exploitation and exploration strategy, effectively dealing with the high geometric complexity and local solution traps. With maximum respect for the original code setup, we modified the source code in MATLAB [3] on

[1] https://github.com/geatpy-dev/geatpy
[2] https://github.com/Will-iam-L/DEA-DYPN
[3] https://www.mathworks.com/matlabcentral/fileexchange//156906-fdb-ea-a-new-routing-algorithm-and-tsp-d-problem-codes?s_tid=srchtitle

fitness calculation to make it applicable to OPs. To keep the number of evaluations similar to other comparison algorithms (i.e., SEGA), the population size and maximum number of generations are set as 250 and 40,000, respectively. Other settings are kept the same as the default.

4) AM: AM is a state-of-the-art deep reinforcement learning method proposed by Kool et al. [22] and has shown excellent performance in solving various types of routing problems. This method demonstrates an ability to dynamically learn to concentrate on pertinent information within the input data, allowing it to effectively tackle challenging routing problems. Based on the default parameters, we use the same training settings as that of the proposed DYPN, that is, training AM with 1,280,000 50-node OP instances for 10 epochs. The batch size is 256 and the total model training time is about 2 hours.

5) Tsili-Greedy: The Tsili-Greedy algorithm is a heuristic method specially designed for the OP [48]. It is known for its simplicity and effectiveness in quickly obtaining near-optimal solutions. The Tsili-Greedy operates greedily, making locally optimal choices at each step to construct a feasible route that maximizes the total collected score within the limited travel distance. In the context of the OP, the Tsili-Greedy algorithm has demonstrated competitive performance and computational efficiency, making it a valuable heuristic approach for solving this specific issue.

V. Experimental Results and Discussion

In this section, we test all the competitor algorithms on the given test cases with sizes of 20, 50, 100, 200, and 500. The performance analysis, significance test, and stability test are conducted according to the optimization results. After that, the complexity analysis, sensitivity analysis, and several ablation experiments are conducted to further validate the proposed method.

A. Model Performance

OP instances with sizes of 20, 50, 100, 200, and 500 are used to test the competitor algorithms. One randomly generated instance is included for each size and is tested for 10 independent runs for each algorithm. The average obtained scores (i.e., AVG), the variance of the obtained scores (i.e., VAR), the average running time (i.e., Time), and the Friedman ranks (i.e., F Rank) are used to evaluate the model performance. The highest average score for each instance is labeled in gray and is set as the baseline for calculating optimality gaps. The gap metric is defined as the relative difference between the objective score obtained by an algorithm and the best-known solution. TABLE II shows the experimental results of the competitor algorithms. Gurobi-30 and Gurobi-300 respectively refer to that the maximum running time of Gurobi is set as 30 and 300 seconds. When Gurobi finds the optimal solution before its maximum running time, it stops searching automatically. Also, the running time of SEGA, FDB-EA, and DEA-DYPN is limited to 300 seconds for fair comparison (slightly exceeding 300s is allowed since these algorithms can only be interrupted after each evolutionary generation).

1) Performance Analysis: According to the test results shown in TABLE II, DEA-DYPN performs the best in most cases, except for the Node-50 case, with a gap of 3% lagging behind Gurobi-300. When the problem size grows, DEA-DYPN demonstrates greater dominance over other competitors. The average Friedman rank (i.e., AVG F Rank) on all test cases validates its advance: DEA-DYPN has an average Friedman rank of 1.7 and ranks best. Gurobi-300 comes in second, followed by the AM, Gurobi-30, FDB-EA, Tsili-Greedy, and SEGA.

Gurobi is one of the most advantaged commercial solvers. It shows powerful optimization capabilities on small-scale problems but is weak for large-scale cases, such as the Node-200 and Node-500 test cases. Representative evolutionary algorithms including SEGA and FDB-EA face a similar issue. The increase in the number of decision variables brings an exponential increase in the search space, and these algorithms can easily fall into local optima on large-scale problems. This is one of the main challenges hindering their application to large-scale complex COPs. Whereas, advanced operator design brings better optimization ability for FDB-EA to some degree, compared with the traditional SEGA.

As for AM, while it is trained on instances with 50 nodes, it does not show significant performance degradation on the Node-20 and Node-100 cases. This is due to the strong generalization performance of the DRL model, which allows it to perform well on test sets similar to the training sets. However, due to the complexity of the OP, AM is difficult to achieve better results, when compared with our method. Tsili-Greedy, one classical greedy heuristic specifically designed for OP, can produce solutions in one second for all instances. However, the obtained solutions are worse than those of Gurobi, AM, and DEA-DYPN in all cases.

In addition to being insensitive to problem size, the proposed DEA-DYPN is also insensitive to OP parameters such as the maximum tour length and the node score distribution. This is because these parameters are not involved in the training and running of the inner DYPN model, but are only used in the external optimizer DEA. This leads to the fact that even if these parameters change a lot, we still do not have to retrain the DYPN model. In contrast, the pure DRL method AM will need to be retrained to get better-optimized performance when facing this situation.

As for the running time, it is obvious that AM and Tsili-Greedy have outstanding advantages. AM benefits from its end-to-end structure and Tsili-Greedy benefits from its simple computational logic. Apart from these, Gurobi runs within its runtime limitations and others require minutes. Within the limited running time (i.e., 300 seconds), DEA-DYPN obtains better solutions in less time compared to SEGA and FDB-EA.

2) Significance Test: To assess the statistical significance of the performance differences observed among the proposed DEA-DYPN and benchmark algorithms, we employed the Friedman test [49] followed by the post-hoc Nemenyi test [50] in this subsection.

The Friedman test, a non-parametric alternative to the repeated measures ANOVA, ranks algorithms for each instance separately, with ranks averaged across all instances. The null

TABLE II
EVALUATION RESULTS OF THE COMPETITOR ALGORITHMS ON THE TEST INSTANCES WITH SIZES OF 20, 50, 100, 200, AND 500. THE BEST VALUES ARE LABELED IN GRAY.

		Gurobi-30	Gurobi-300	SEGA	FDB-EA	AM	Tsili-Greedy	DEA-DYPN
Node-20	AVG	5.83	5.83	5.80	5.83	5.79	5.79	5.83
	Gap	0%	0%	1%	0%	1%	1%	0%
	VAR	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Time(s)	9	9	217	139	2	1	47
	F Rank	2.50	2.50	5.00	2.50	6.50	6.50	2.50
Node-50	AVG	14.31	14.37	10.35	12.10	13.60	11.88	13.92
	Gap	0%	0%	28%	16%	5%	17%	3%
	VAR	0.00	0.00	0.65	1.12	0.00	0.00	0.00
	Time(s)	30	300	301	204	2	1	95
	F Rank	2.00	1.00	7.00	5.00	4.00	6.00	3.00
Node-100	AVG	31.13	32.10	18.69	20.98	28.37	18.53	32.77
	Gap	5%	2%	43%	36%	13%	43%	0%
	VAR	0.00	0.00	1.52	1.23	0.00	0.00	0.00
	Time(s)	30	300	306	290	3	1	226
	F Rank	3.00	2.00	6.00	5.00	4.00	7.00	1.00
Node-200	AVG	31.37	44.38	25.96	29.78	45.18	37.95	49.71
	Gap	37%	11%	48%	40%	9%	24%	0%
	VAR	0.00	0.00	7.74	5.29	0.00	0.00	0.10
	Time(s)	34	305	302	310	5	1	308
	F Rank	5.00	3.00	7.00	6.00	2.00	4.00	1.00
Node-500	AVG	37.96	76.67	34.29	41.26	95.12	93.78	111.04
	Gap	66%	31%	69%	63%	14%	16%	0%
	VAR	0.00	0.00	26.87	12.05	0.00	0.00	1.75
	Time(s)	50	321	311	309	9	1	315
	F Rank	7.00	4.00	6.00	5.00	2.00	3.00	1.00
AVG VAR		0.00	0.00	7.36	3.94	0.00	0.00	0.37
AVG F Rank		3.90	2.50	6.20	4.70	3.70	5.30	1.70

hypothesis (H_0) assumes that all algorithms are equivalent in terms of their performance metrics. The Friedman statistic, calculated based on the Friedman ranks, is compared against the critical value from the chi-squared distribution with $k-1$ degrees of freedom, where k is the number of algorithms and is 7 in our study. The derived P-value is 0.04 (< 0.05), which rejects the null hypothesis. This suggests that at least one algorithm performs significantly differently from the others.

After that, the post-hoc Nemenyi test is conducted to further compare algorithmic differences. The post-hoc Nemenyi test is a statistical method used to determine which pairs of algorithms exhibit significant differences in performance by comparing their average ranks across multiple datasets. The derived P-values of pairwise comparisons are visualized in Fig. 3. The lower the P-value, the more significant the difference between the algorithms. This result reveals that DEA-DYPN significantly differs from SEGA, Tsili-Greedy, and FDB-EA. In contrast, DEA-DYPN shows a relatively small difference with Gurobi-300. Combining with the experimental results shown in TABLE II, the performance advantage of DEA-DYPN over others is validated statistically.

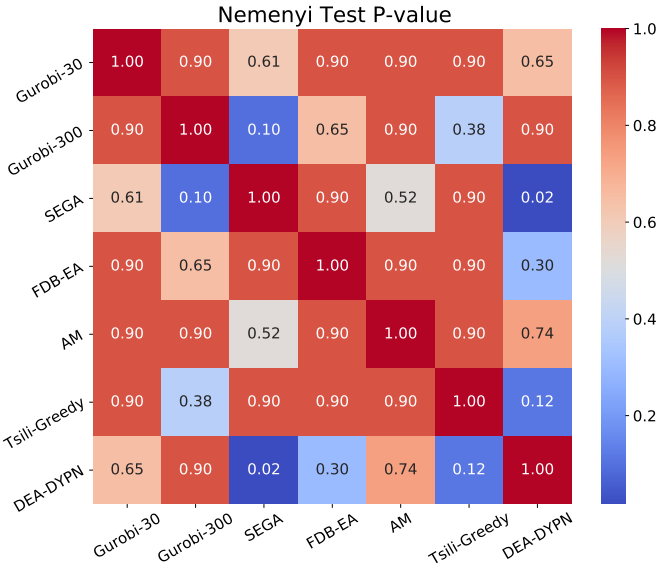


Fig. 3. The heatmap of the post-hoc Nemenyi test p-values among competitor algorithms.

3) Stability Analysis: The algorithmic optimization stability is also an important evaluation metric [51], and is defined as the variance of the obtained scores in 10 independent runs on a fixed test case in this study. TABLE II contains the variances of each algorithm on each test case, as labeled as VAR. For each algorithm, the average VAR value on all test cases is called AVG VAR, which shows the optimization stability. Since no random search operator is contained, Gurobi-30, Gurobi-300, AM, and Tsili-Greedy always generate the same solutions on a fixed test case, no matter how many times they run. In contrast, SEGA, FDB-EA, and DEA-DYPN may generate different solutions in different runs, due to their perturbation operators. The experimental results show that DEA-DYPN has much better stability than SEGA and FDB-EA.

Fig. 4 shows the raincloud plots of the solutions obtained by the compared algorithms. The raincloud plot is essentially a combination of a violin plot and a box plot, which can display the shape of the distribution, median, quartiles, and extreme values of the data simultaneously. The red dots connected by lines in these raincloud plots indicate the average values. These

visualized solutions further validate the conclusions we had previously reached.

4) Performance Conclusion: Overall, the DEA-DYPN can well balance the optimization accuracy and efficiency on most of the test instances, due to its double-layer optimization framework and effective optimization module design. It performs comparably to Gurobi on small-scale problems (i.e., Node-20 and Node-50), and shows a lead on larger scales (i.e., Node-100, Node-200, and Node-500). The problem decomposition-based idea reduces the model complexity effectively, making this approach suited for solving large-scale problems. Moreover, due to the end-to-end feed-forward design of DYPN and the matrix arithmetic form, DYPN can immediately plan paths for the entire population at the same time. It is worth noting that such a double-layer optimization approach differs from neural network surrogate models, which do not produce concrete solutions but estimate the objective function acting like a black box. In contrast, DYPN produces solutions to the subproblem directly and calculates its objective function based on exact mathematical operations, which is much more intuitive.

B. Complexity Analysis

The time complexity of the proposed DEA-DYPN, traditional EA, and AM is analyzed according to their algorithmic modules shown in Fig. 1. We reset the relevant notations in this subsection for more convenient interpretation: setting the maximum number of generations as g, the population size as m, the problem size (i.e. number of nodes) as n, and the hidden layer dimension as d. TABLE III lists the algorithmic time complexity of all modules.

The greedy population initialization in DEA-DYPN, as a distinctive feature of our approach, has a complexity of $O(m \times n^2)$. This is due to the need to evaluate each remaining node for inclusion in each solution. The crossover and mutation operations perform with complexities of $O(g \times m \times n)$, typical for evolutionary algorithms. The environmental selection operator has a complexity of $O(g \times m^2)$, due to the need to evaluate and rank individuals for g generations.

The path planning module DYPN has an encoder with a complexity of $O(g \times n \times d)$ and a decoder with a complexity of $O(g \times n^2 \times d)$. Therefore, the complexity for planning paths is $O(g \times n^2 \times d)$. This computation process allows paths to be planned for the entire population at once due to the matrix computational design and is therefore independent of the population size m. It is worth noting that, the standard AM only has to run the network feed-forward once, and therefore has a complexity of $O(n^2 \times d)$.

According to TABLE III, it is clear that DEA-DYPN has the highest time complexity when operating parameters are set the same. However, algorithmic complexity analysis without incorporating optimization quality is meaningless. An advanced operator design is capable of significantly improving search efficiency and quality, that is, it requires much less population size m and much fewer evolutionary generations g to obtain a better solution. This is why the proposed DEA-DYPN only evolves 256 individuals for 400 generations but outperforms SEGA and FDB-EA, which evolve 256 individuals for 40,000 generations. Thus, in terms of the computational resources required to achieve the same optimization result, it is clear that DEA-DYPN is more efficient, even though it has a higher complexity in theory. On the other hand, while the computational complexity of AM is low, it is difficult to generate high-quality solutions to complex problems in a single feed-forward process due to the limited network structure.

The advantage of DEA-DYPN in optimization efficiency can also be explained by analyzing the search space. Given the parallelizable nature of neural computations, DYPN can efficiently plan paths with GPU acceleration. The DEA module only needs to select nodes, with a search space size of 2^n. In contrast, the search space size of a general EA for solving an OP is $\sum_{i=1}^{n} C_n^i \cdot i!$, which is much larger. This is the core difficulty for traditional EAs to solve large-scale OP problems.

C. Learning Ability of Dynamic Pointer Network

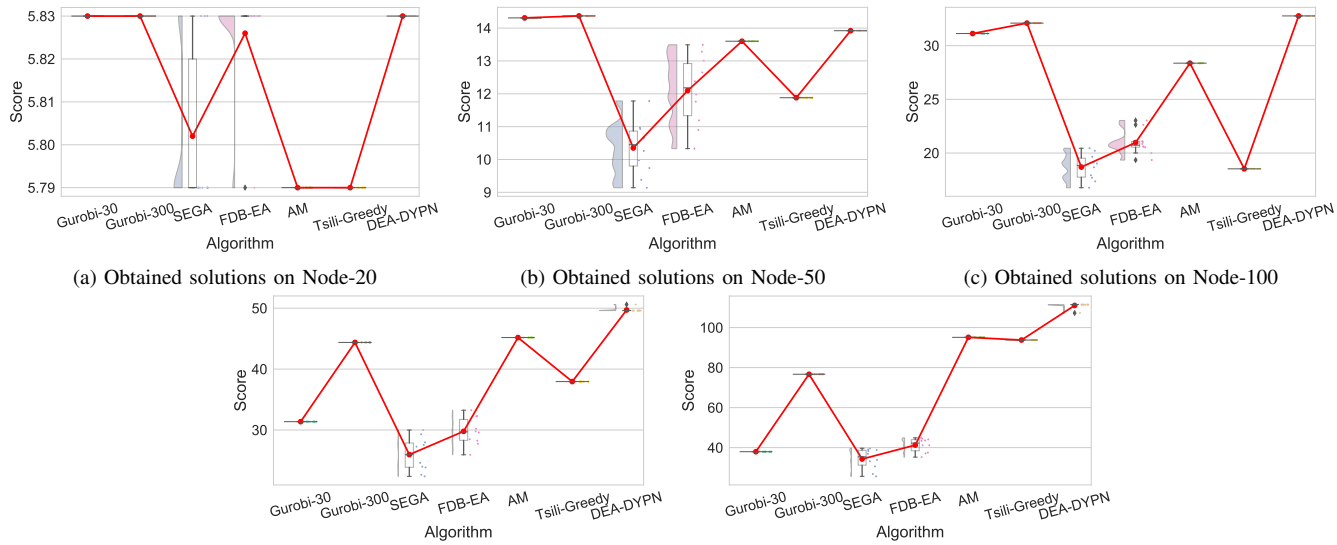
To validate the effectiveness of the proposed dynamic embedding mechanism in DYPN, this subsection visualized the training cost of DYPN and the traditional PN [4], as shown in Fig. 5. Their training parameters are set as the same, including the problem size of 20, the training instance size of 1,280,000, the batch size of 1024, and the training epoch size of 10. The training cost is defined as the total tour length of solutions. In Fig. 5, the curves for the initial 2000 batches and the last 2000 batches are separated on the right to show clearer gaps. As can be seen, DYPN outperforms PN with a higher convergence speed but a lower cost. Owing to the extra distance information, the agent makes decisions with heuristic guidance in DYPN.

Fig. 6 shows the performance of the pre-trained DYPN model (trained on 50 nodes) in planning TSP paths with problem sizes of 20, 50, and 100. Nodes in these cases are randomly distributed within a unit square. The average tour lengths of the validation results are 3.95, 6.08, and 8.51, respectively. As can be seen, the route planning schemes shown in this figure are intuitively reasonable and outstanding.

D. Hyperparametric Sensitivity Analysis

This subsection makes a sensitivity analysis of the key hyperparameter settings of Adam, including the learning rate and the dropout rate. Setting the learning rate as 5e-6, 5e-5, 5e-4, 5e-3, and 5e-2, and setting the dropout rate as 0.01, 0.05, 0.1, 0.15, and 0.2, we independently train the DYPN model on 50-node TSP instances. The batch size is set as 1024, the epoch number is set as 10, and number of training instances in each epoch is set as 128,000. These pre-trained models are validated on 100 50-node randomly generated test instances. The average route length is used as the evaluation metric. Fig. 7 presents the validation result. As can be seen, the learning rate of 5e-3 performs the best, while the dropout rate has few effects. Fig. 8 visualizes the detailed learning process of the models with a dropout rate of 0.05. This result is consistent with our general knowledge: too large a learning rate leads to

[4]https://github.com/mveres01/pytorch-drl4vrp



(a) Obtained solutions on Node-20 (b) Obtained solutions on Node-50 (c) Obtained solutions on Node-100

(d) Obtained solutions on Node-200 (e) Obtained solutions on Node-500

Fig. 4. Raincloud plots of the solutions obtained by the competitor algorithms. Test instances sizing 20, 50, 100, 200, and 500 are included. These plots show the distribution of the obtained solutions over the score evaluation metrics.

TABLE III
TIME-COMPLEXITY ANALYSIS OF THE PROPOSED DEA-DYPN, TRADITIONAL EA, AND AM.

DEA-DYPN		EA		AM	
Module	**Complexity**	**Module**	**Complexity**	**Module**	**Complexity**
(1) Population Initialization	$O(m \times n^2)$	(1) Population Initialization	$O(m)$	(1) Path Planning	$O(n^2 \times d)$
(2) Path Planning	$O(g \times n^2 \times d)$	(2) Crossover and Mutation	$O(g \times m \times n)$		
(3) Crossover and Mutation	$O(g \times m \times n)$	(3) Environmental Selection	$O(g \times m^2)$		
(4) Environmental Selection	$O(g \times m^2)$	(4) Output Solution	$O(1)$		
(5) Output Solution	$O(1)$				
Overall Complexity	$O(m \times n^2 + g \times n^2 \times d + g \times m \times n + g \times m^2)$	**Overall Complexity**	$O(g \times m \times n + g \times m^2)$	**Overall Complexity**	$O(n^2 \times d)$

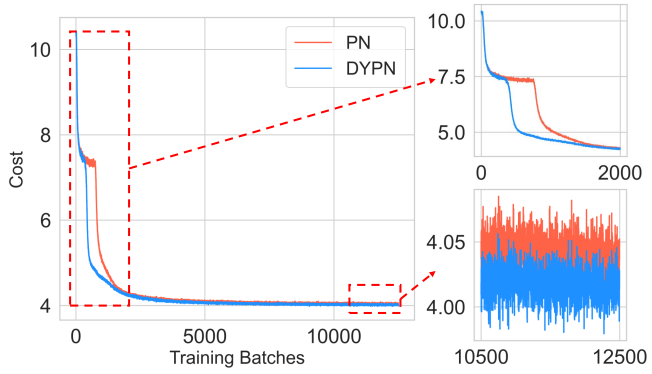


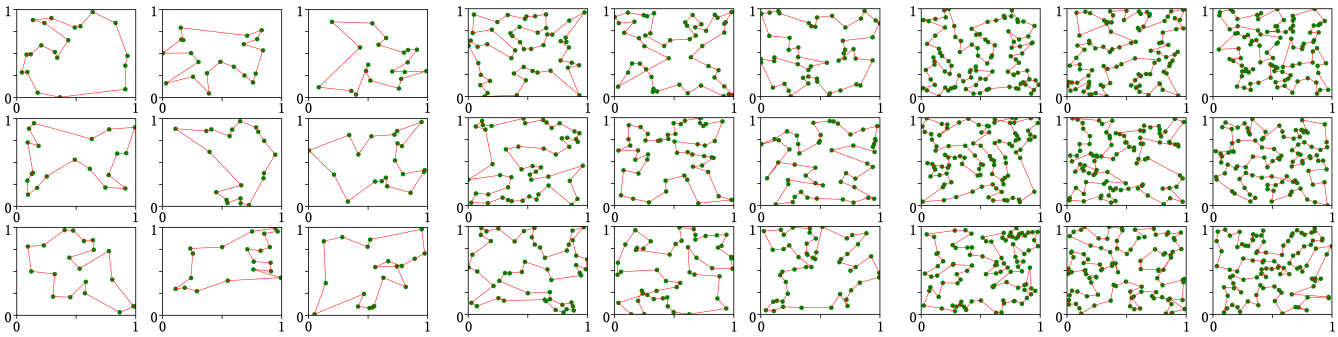
Fig. 5. Cost changes along with the number of batches in the training stage for DYPN and PN.

difficult convergence, while too small a learning rate leads to inefficient learning. On the other hand, the dropout rate is used to prevent model overfitting. A large dropout rate generally leads to strong generalization but inefficient learning and a small dropout rate is the opposite. In conjunction with our experimental results, we end up setting the learning rate and dropout rate as 5e-3 and 0.05, respectively.

E. Ablation Experiments on Diversity Evolutionary Algorithm

This subsection conducts ablation experiments of the introduced operators on DEA, aiming to verify the algorithmic effect enhancement brought by each operator, individually. Ablation experiments of three proposed operators are respectively discussed, including the greedy population initialization heuristic, the population restart mechanism, and the fitness-sharing selection strategy. During each of these ablation experiments, in addition to the single operator to be discussed, all other algorithm designs as well as the parameter settings are the same. In these three ablation experiments, the greedy population initialization heuristic is replaced by a random population initialization method (therefore call the new approach DEA-DYPN/greedy), the population restart mechanism is replaced by none (therefore call the new approach DEA-DYPN/Restart), and the fitness-sharing selection strategy is replaced by a general fitness-ranking selection operator (therefore call the new approach DEA-DYPN/FitShare). We re-run these new approaches on the 20-node, 50-node, and 100-node test instances. 10 randomly generated instances are included for each problem size. The average obtained scores and the running time are used as the evaluation criteria for algorithms. TABLE IV lists the experimental results.

As can be seen, compared to the complete DEA-DYPN method, the other ablated approaches (i.e.,



(a) Paths on **Node-20** TSP cases (b) Paths on **Node-50** TSP cases (c) Paths on **Node-100** TSP cases

Fig. 6. Paths planned by the pre-trained DYPN model on TSP cases with sizes of 20, 50, and 100.

TABLE IV
EVALUATION RESULTS OF THE ABLATION EXPERIMENTS FOR DEA. THE GREEDY POPULATION INITIALIZATION HEURISTIC, THE POPULATION RESTART MECHANISM AND THE FITNESS-SHARING SELECTION STRATEGY ARE TESTED INDIVIDUALLY. THE BEST VALUES ARE LABELED IN GRAY.

	Node-20					Node-50					Node-100					AVG F Rank
	AVG	Gap	VAR	Time(s)	F Rank	AVG	Gap	VAR	Time(s)	F Rank	AVG	Gap	VAR	Time(s)	F Rank	
DEA-DYPN	5.59	0%	1.24	47	1.00	15.28	0%	4.42	95	1.00	30.01	0%	6.13	226	1.00	1.00
DEA-DYPN/Greedy	5.51	1%	1.24	38	2.00	15.21	0%	4.91	77	3.00	29.40	2%	5.48	190	3.00	2.67
DEA-DYPN/Restart	5.52	1%	1.13	45	3.00	15.25	0%	4.65	88	2.00	30.00	0%	5.62	209	2.00	2.33
DEA-DYPN/FitShare	5.48	2%	1.00	26	4.00	15.04	2%	5.29	56	4.00	29.50	2%	6.72	148	4.00	4.00

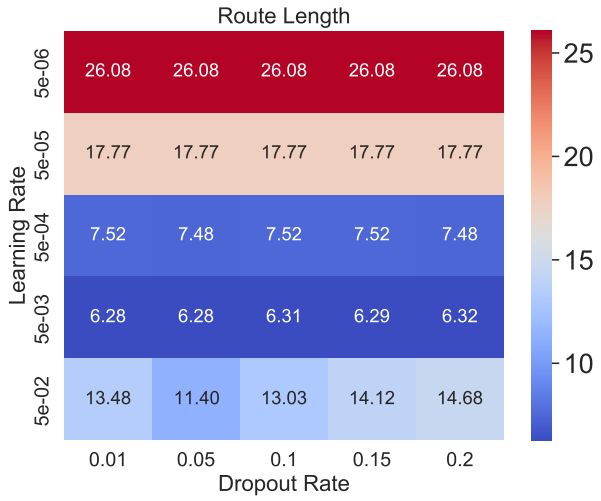


Fig. 7. The performance of DYPN with different learning rates and dropout rates.

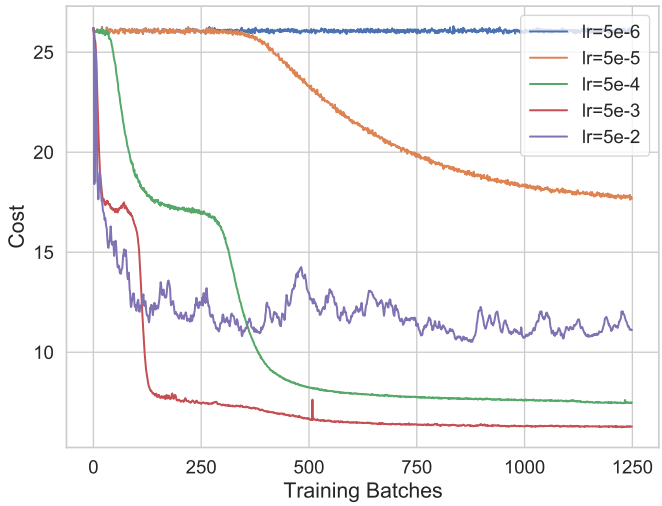
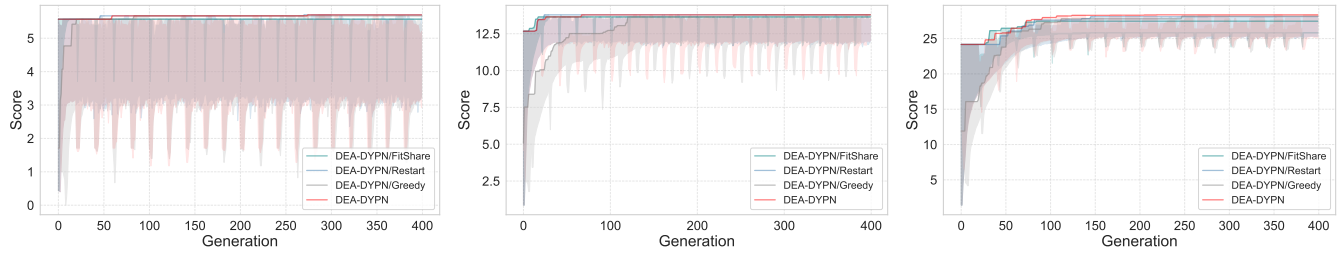


Fig. 8. The training costs of the models with a dropout rate of 0.05 and different learning rates.

DEA-DYPN/Greedy, DEA-DYPN/Restart, and DEA-DYPN/FitShare) all have varying degrees of performance degradation. Such results verify the superiority of the proposed operators in DEA, especially for the greedy population initialization heuristic and the fitness-sharing selection strategy. Of course, with the introduction of more complex optimization operators, DEA takes longer to solve the problem.

Fig. 9 visualized the evolutionary process of DEA-DYPN and its ablated variants respectively on the Node-20, Node-50, and Node-100 cases. For the problem of each size, one is randomly selected from the given 10 test instances to show the evolutionary process. Lines in these figures represent the optimal solutions during evolution, and the shaded area represents the region contained in the optimal and the worst solutions of each newly generated population.

Many properties of the compared algorithms can be intuitively found from Fig. 9. First, DEA-DYPN, as shown in red in these figures, has the fastest convergence rate and can obtain the best solutions. Benefits from the utilization of various diversity operators, the population of DEA-DYPN distributes over a wider area than other ablated variants, as the red area in the figures. In contrast, the population distribution area of DEA-DYPN/FitShare and DEA-DYPN/Restart is narrow. Secondly, it is obvious that, without an effective population initialization method, DEA-DYPN/Greedy has a weak initial population, resulting in a slow convergence. This further validates the effectiveness of the proposed greedy population



(a) Evolutionary process on Node-20 (b) Evolutionary process on Node-50 (c) Evolutionary process on Node-100

Fig. 9. Evolutionary process of the proposed DEA-DYPN and its ablated variants (i.e., DEA-DYPN/Greedy, DEA-DYPN/Restart, and DEA-DYPN/FitShare). Three test instances respectively with the sizes of 20, 50, and 100 and included. While the lines represent the optimal solutions during evolution, the shaded area indicates the region contained in the optimal and worst solutions of each newly generated population.

initialization heuristic. Thirdly, the use of the population restart mechanism brings bigger search space when the population stagnates. The loss of this mechanism makes DEA-DYPN/Restart hard to jump out of local optima. Overall, the ablation experiments have effectively verified the superiority of the three proposed optimization operators.

VI. Conclusion

In this study, we use a problem decomposition-based double-layer optimization framework DEA-DYPN to solve OPs. By decomposing an OP into a much simpler KP and TSP, we are better able to deal with large-scale OP cases. A diversity evolutionary algorithm is proposed as the external optimizer to solve the KP. By introducing several effective operators including a greedy population initialization heuristic, an elite strategy, a population restart mechanism, and a fitness-sharing selection strategy, DEA can effectively handle KP cases with hundreds of nodes. As for the inner optimizer, we train a dynamic pointer network offline and run it online during the evolution. The end-to-end mode and the parallel computational design make DYPN being able to plan paths for the whole population in real time. Such a design greatly reduces the difficulty of solving the problem, achieving a better balance of optimization accuracy and efficiency. Computational experiments with sizes from 20 to 500 are conducted. Representative algorithms including Gurobi, SEGA, FDB-EA, AM, and Tsili-Greedy are in comparison. The comparative results, significance test, stability analysis, complexity analysis, sensitivity analysis, and ablation experiments validate the effectiveness and generalization ability of the proposed DEA-DYPN.

Combining EAs with DRL methods to solve COPs is an interesting and promising research direction. Although many researchers are now exploring this line of research, few research results are generalizable. How to improve the generalization of such hybrid algorithms may be a point of interest in the future. The variety of interactions brings more possibilities for such hybrid algorithms.

References

[1] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and N. Vathis, "Heuristics for the time dependent team orienteering problem: Application to tourist route planning," *Computers & Operations Research*, vol. 62, pp. 36–50, 2015.

[2] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011.

[3] F. Mufalli, R. Batta, and R. Nagi, "Simultaneous sensor selection and routing of unmanned aerial vehicles for complex mission plans," *Computers & Operations Research*, vol. 39, no. 11, pp. 2787–2799, 2012.

[4] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, and R. Siegwart, "A uav system for inspection of industrial facilities," in *2013 IEEE Aerospace Conference*. IEEE, 2013, pp. 1–8.

[5] M. Chernigovskaya, A. Kharitonov, and K. Turowski, "A recent publications survey on reinforcement learning for selecting parameters of meta-heuristic and machine learning algorithms," in *International Conference on Cloud Computing and Services Science*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258402119

[6] O. Udomkasemsub, B. Sirinaovakul, and T. Achalakul, "Phh: Policy-based hyper-heuristic with reinforcement learning," *IEEE Access*, vol. 11, pp. 52 026–52 049, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258818158

[7] G. Laporte and S. Martello, "The selective travelling salesman problem," *Discrete applied mathematics*, vol. 26, no. 2-3, pp. 193–207, 1990.

[8] R. Ramesh, Y.-S. Yoon, and M. H. Karwan, "An optimal algorithm for the orienteering tour problem," *ORSA Journal on Computing*, vol. 4, no. 2, pp. 155–165, 1992.

[9] A. C. Leifer and M. B. Rosenwein, "Strong linear programming relaxations for the orienteering problem," *European Journal of Operational Research*, vol. 73, no. 3, pp. 517–523, 1994.

[10] M. Fischetti, J. J. S. Gonzalez, and P. Toth, "Solving the orienteering problem through branch-and-cut," *INFORMS Journal on Computing*, vol. 10, no. 2, pp. 133–148, 1998.

[11] M. Gendreau, G. Laporte, and F. Semet, "A branch-and-cut algorithm for the undirected selective traveling salesman problem," *Networks: An International Journal*, vol. 32, no. 4, pp. 263–273, 1998.

[12] A. Gunawan, H. C. Lau, P. Vansteenwegen, and K. Lu, "Well-tuned algorithms for the team orienteering problem with time windows," *Journal of the Operational Research Society*, vol. 68, no. 8, pp. 861–876, 2017.

[13] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden, "Iterated local search for the team orienteering problem with time windows," *Computers & Operations Research*, vol. 36, no. 12, pp. 3281–3290, 2009.

[14] G. Kobeaga, M. Merino, and J. A. Lozano, "An efficient evolutionary algorithm for the orienteering problem," *Computers & Operations Research*, vol. 90, pp. 42–59, 2018.

[15] R. Pěnička, J. Faigl, and M. Saska, "Variable neighborhood search for the set orienteering problem and its application to other orienteering problem variants," *European Journal of Operational Research*, vol. 276, no. 3, pp. 816–825, 2019.

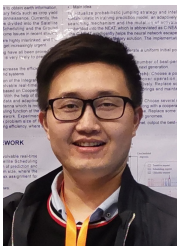
[16] X. Chou, L. M. Gambardella, and R. Montemanni, "A tabu search algorithm for the probabilistic orienteering problem," *Computers & Operations Research*, vol. 126, p. 105107, 2021.

[17] Y. Amarouche, R. N. Guibadj, E. Chaalal, and A. Moukrim, "Effective neighborhood search with optimal splitting and adaptive memory for the team orienteering problem with time windows," *Computers & Operations Research*, vol. 123, p. 105039, 2020.

[18] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[19] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural

combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.
[20] M. Nazari, A. Oroojlooy, L. V. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *Neural Information Processing Systems*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:46892983
[21] K. Li, T. Zhang, and R. Wang, "Deep reinforcement learning for multiobjective optimization," *IEEE transactions on cybernetics*, vol. 51, no. 6, pp. 3103–3114, 2020.
[22] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:59608816
[23] Y. Xu, M. Fang, L. Chen, G. Xu, Y. Du, and C. Zhang, "Reinforcement learning with multiple relational attention for solving vehicle routing problems," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 11 107–11 120, 2021.
[24] H. Lu, X. Zhang, and S. Yang, "A learning-based iterative method for solving vehicle routing problems," in *International conference on learning representations*, 2019.
[25] R. Qi, J.-q. Li, J. Wang, H. Jin, and Y.-y. Han, "Qmoea: A q-learning-based multiobjective evolutionary algorithm for solving time-dependent green vehicle routing problems with time windows," *Information Sciences*, vol. 608, pp. 178–201, 2022.
[26] F. Zhang, R. Li, and W. Gong, "Deep reinforcement learning-based memetic algorithm for energy-aware flexible job shop scheduling with multi-agv," *Computers & Industrial Engineering*, vol. 189, p. 109917, 2024.
[27] T. C. Bora, V. C. Mariani, and L. dos Santos Coelho, "Multi-objective optimization of the environmental-economic dispatch with reinforcement learning based on non-dominated sorting genetic algorithm," *Applied Thermal Engineering*, vol. 146, pp. 688–700, 2019.
[28] T. C. Bora, L. Lebensztajn, and L. D. S. Coelho, "Non-dominated sorting genetic algorithm based on reinforcement learning to optimization of broad-band reflector antennas satellite," *IEEE transactions on magnetics*, vol. 48, no. 2, pp. 767–770, 2012.
[29] J. Kallestad, R. Hasibi, A. Hemmati, and K. Sörensen, "A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems," *European Journal of Operational Research*, vol. 309, no. 1, pp. 446–468, 2023.
[30] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 9, pp. 5057–5069, 2021.
[31] P. Liang, Y. Chen, Y. Sun, Y. Huang, and W. Li, "An information entropy-driven evolutionary algorithm based on reinforcement learning for many-objective optimization," *Expert Systems with Applications*, vol. 238, p. 122164, 2024.
[32] G. Watkins, G. Montana, and J. Branke, "Generating a graph colouring heuristic with deep q-learning and graph neural networks," in *International Conference on Learning and Intelligent Optimization*. Springer, 2023, pp. 491–505.
[33] W. Liu, R. Wang, T. Zhang, K. Li, W. Li, H. Ishibuchi, and X. Liao, "Hybridization of evolutionary algorithm and deep reinforcement learning for multiobjective orienteering optimization," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 5, pp. 1260–1274, 2023.
[34] Y. Song, Y. Wu, Y. Guo, R. Yan, P. N. Suganthan, Y. Zhang, W. Pedrycz, S. Das, R. Mallipeddi, O. S. Ajani *et al.*, "Reinforcement learning-assisted evolutionary algorithm: A survey and research opportunities," *Swarm and Evolutionary Computation*, vol. 86, p. 101517, 2024.
[35] Q. Zhu, X. Wu, Q. Lin, L. Ma, J. Li, Z. Ming, and J. Chen, "A survey on evolutionary reinforcement learning algorithms," *Neurocomputing*, vol. 556, p. 126628, 2023.
[36] S. Qi, J. Zou, S. Yang, Y. Jin, J. Zheng, and X. Yang, "A self-exploratory competitive swarm optimization algorithm for large-scale multiobjective optimization," *Information sciences*, vol. 609, pp. 1601–1620, 2022.
[37] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 69–93.
[38] H. T. Kahraman, S. Aras, and E. Gedikli, "Fitness-distance balance (fdb): a new selection method for meta-heuristic search algorithms," *Knowledge-Based Systems*, vol. 190, p. 105169, 2020.
[39] H. T. Kahraman, M. Katı, S. Aras, and D. A. Taşci, "Development of the natural survivor method (nsm) for designing an updating mechanism in metaheuristic search algorithms," *Engineering Applications of Artificial Intelligence*, vol. 122, p. 106121, 2023.
[40] B. Ozkaya, H. T. Kahraman, S. Duman, and U. Guvenc, "Fitness-distance-constraint (fdc) based guide selection method for constrained optimization problems," *Applied Soft Computing*, vol. 144, p. 110479, 2023.
[41] K. Cho, B. van Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing*, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:5590763
[42] R. Gama and H. L. Fernandes, "A reinforcement learning approach to the orienteering problem with time windows," *Computers & Operations Research*, vol. 133, p. 105357, 2021.
[43] J. Lehr, J. Philipps, V. N. Hoang, D. von Wrangel, and J. Krüger, "Supervised learning vs. unsupervised learning: A comparison for optical inspection applications in quality control," *IOP Conference Series: Materials Science and Engineering*, vol. 1140, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235293338
[44] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.
[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106
[46] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2021.
[47] C. Yılmaz, E. Cengiz, and H. T. Kahraman, "A new evolutionary optimization algorithm with hybrid guidance mechanism for truck-multi drone delivery system," *Expert Systems with Applications*, vol. 245, p. 123115, 2024.
[48] T. Tsiligirides, "Heuristic methods applied to orienteering," *Journal of the Operational Research Society*, vol. 35, pp. 797–809, 1984.
[49] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.
[50] P. B. Nemenyi, *Distribution-free multiple comparisons.* Princeton University, 1963.
[51] H. T. Öztürk and H. T. Kahraman, "Meta-heuristic search algorithms in truss optimization: Research on stability and complexity analyses," *Applied Soft Computing*, vol. 145, p. 110573, 2023.



Rui Wang (Senior Member, IEEE) received his Bachelor's degree from the National University of Defense Technology, P.R. China in 2008, and the Doctor degree from the University of Sheffield, U.K in 2013. Currently, he is with the National University of Defense Technology. His current research interest includes evolutionary computation, multi-objective optimization, and the development of algorithms applicable in practice.

Dr. Wang received the Operational Research Society Ph.D. Prize in 2016, and the National Science Fund for Outstanding Young Scholars in 2021. He serves as Associate Editors of several leading journals such as IEEE Trans on Evolutionary Computation, Swarm and Evolutionary Computation, and Expert Systems with Applications.

Wei Liu received the B.S. and M.S. degrees in 2020 and 2022, respectively, from the National University of Defense Technology, Changsha, China, where he is currently pursuing the Ph.D. degree in management science and engineering. His main research directions are deep reinforcement learning, evolution algorithms, and multiobjective optimization.

Kaiwen Li received the B.S., M.S., and Ph.D degrees from National University of Defense Technology (NUDT), Changsha, China, in 2012, 2016 and 2018. He is an associate professor with the College of Systems Engineering, NUDT. His research interests include prediction techniques, multiobjective optimization, reinforcement learning, data mining, and optimization methods on the Energy Internet.