Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Deep reinforcement learning for transportation network combinatorial optimization: A survey

Qi Wang [a], Chunlei Tang (PhD) [b,*]

[a] Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, Shanghai, China
[b] Division of General Internal Medicine and Primary Care, Brigham and Women's Hospital, Harvard Medical School, Boston, MA 02120, USA

ARTICLE INFO

Article history:
Received 16 January 2021
Received in revised form 18 September 2021
Accepted 20 September 2021
Available online 27 September 2021

Keywords:
Combinatorial optimization
Deep learning
Reinforcement learning
Graph neural network

ABSTRACT

Traveling salesman and vehicle routing problems with their variants, as classic combinatorial optimization problems, have attracted considerable attention for decades of their theoretical and practical value. Many classic algorithms have been proposed, for example, exact algorithms, heuristic algorithms, solution solvers, etc. Still, due to their complexity, even the most advanced traditional methods require too much computational time or are not well-defined mathematically; algorithm-based decision-making is no exception. Also, these methods cannot be generalized to a larger scale or other similar problems. With the latest developments in machine and deep learning, people believe it is feasible to apply reinforcement learning and other technologies in the decision-making or heuristic for learning combinatorial optimization. In this paper, we first gave an overview on how combinate deep reinforcement learning for the NP-hard combinatorial optimization, emphasizing general optimization problems as data points and exploring the relevant distribution of data used for learning in a given task. We next reviewed state-of-the-art learning techniques related to combinational optimization problems on graphs. Then, we summarized the experimental methods of using reinforcement learning to solve combinatorial optimization problems and analyzed the performance comparison of different algorithms. Lastly, we sorted out the challenges encountered by deep reinforcement learning in solving combinatorial optimization problems and future research directions.

1. Introduction

Operations research (OR) emerged during World War II as an important means of adapting mathematics and computer science to assist civilian and military leaders make decisions. OR techniques are now extended to a wide range of industries, including, but not limited to, transportation, supply chains, energy, finance, and scheduling. This research mainly focuses on integer-constrained optimization problems [1], involving optimization problems with integer or binary variables (decision variables).

One of the most famous NP-hard problems [2], the traveling salesman problem (TSP) [2,3], proposes the following question: "Given a list of cities and the distance between each pair of cities, what is the shortest possible route to visit each city and return to the original city?". The vehicle routing problem (VRP) [4–6] says: "One (or more) vehicle is responsible for the goods delivered to multiple clients when the vehicle must be returned to the goods warehouse. The goal is to optimize a set (or sets) of routes where all routes start and end on a given node (warehouse). To obtain the greatest possible returns, it often requires a vehicle to minimize the total distance and the average service time, and it is usually accompanied by constraints such as vehicle load limit, mileage limit, etc.". TSP is a simple, particular case of VRP because VRP has more optimization objectives and constraints [7] than TSP. Due to its highly structured nature, a similar NP-hard problem can be determined by a sequential decision task on the graph. What we do is search for the space of various permutations and combinations to find an optimal node sequence (tour), which has the most negligible total edge weight (tour length).

Traditionally, algorithms for solving this combinatorial optimization (CO) problem can be divided into exact algorithms that are guaranteed to find the optimal solution and heuristic algorithms that weigh the computational cost [8,9]. However, exact algorithms can use heuristic algorithms internally and vice versa. Heuristics are usually expressed in the form of rules, which can be interpreted as policies. Nevertheless, offering a fast and reliable solution is still a challenging task. Even if there are only a few hundred client nodes, it is sometimes difficult to resolve this problem computationally. In the absence of any artificial reasoning, the latest research trend makes machine learning a compelling choice, and it may become a significant milestone in solving such problems.

* Corresponding author.
E-mail addresses: 17110240039@fudan.edu.cn (Q. Wang), towne@fudan.edu.cn (C. Tang).

https://doi.org/10.1016/j.knosys.2021.107526
0950-7051/© 2021 Elsevier B.V. All rights reserved.

Machine learning (including deep learning and reinforcement learning), which focuses on performing tasks given some (limited and often noisy) data, has replaced humans as algorithmic engineers to solve various problems. It is well suited for natural signals with no clear mathematical formula because the actual data distribution is analytically unknown. Machine learning and CO are closely related, particularly in optimizing to minimize errors between prediction and target, because they can overlap or complement each other in certain areas. From the point of view of CO problems, there are two fundamental motivations for applying machine learning. First, the researcher assumes expert knowledge of optimization algorithms but wants to replace some heavy calculations with quick approximations. Learning can establish such approximations in a general way (without deriving new explicit algorithms). Secondly, sometimes expert knowledge may be insufficient, or some algorithmic decisions may be unsatisfactory, so we need to explore the space of these decisions and learn the best behavior (policy) from them to improve the technological level. Although machine learning is approximate, the relevant work collated in this paper can prove that integrating different parts or components of machine learning does not reduce the overall feasibility and theoretical guarantee of optimality.

Ten years ago, computer vision algorithms used hand-crafted features, but today the features can be learned end-to-end by deep neural networks (DNNs) [10]. When applied to a high-dimensional space with many data points, deep learning has apparent advantages due to its predictive ability. Reinforcement learning enables algorithms to learn decisions for reasoning by interacting with the environment [11]. Heuristic policies can be parameterized by DNNs and trained by RL to obtain more robust algorithms for many different CO problems. We want to train models that can make decisions to solve real problems. Those models must learn to choose appropriate solutions for a problem from the vast potential solutions of the combinatorial space.

2. Model description and problem definition

Given a set of transport requirements and a set of vehicles, we aim to plan a route for the vehicles to execute all of the transport requests by the vehicles at the lowest cost. We need to determine which transport requests to fulfill in which order so that it is feasible to traverse all vehicle routes [5,12] (Fig. 1).

The Capacitated Vehicle Routing Problem (CVRP) [13] is the most studied version of the VRP because it has greater academic significance and practical application value. We will start with this basic variant of VRP with a concrete example that elaborates on the VRP series.

2.1. Problem description

In CVRP, a transportation request includes delivery from a single warehouse (denoted as point 0) to a given set of another n points (often referred to as customers) and the total number of customers is $N = \{1, 2, \dots, n\}$. The customer's demand is the quantity that must be delivered to the customer ($i \in N$) (for example, the weight of the goods to be delivered), which is given by the scalar quantity $q_i \geq 0$. Assume that the fleet $K = \{1, 2, \dots, |K|\}$ is homogeneous, which means that $|K|$ vehicles can be used in the warehouse, all vehicles have the same capacity $Q > 0$, and their costs are the same. The vehicles serving a subset of customers $S \subseteq N$ start in the warehouse, then travel to each customer i, and finally return to the warehouse. The cost of the vehicle from i to j is λ_{ij}.

We can apply undirected graphs or directed graphs to construct the problem described above. Let $V = \{0\} \cup N = \{1, 2, \dots, n\}$ be a collection of nodes. In the case of symmetry (undirected graph), where the cost from i to j is equivalent to the cost from j to i, the graph $G = (V, E)$ is complete, and the edge set $E = \{e = \{i, j\} = \{j, i\}: i, j \in V, i \neq j\}$ and edge cost are both λ_{ij}. Otherwise, if there is at least a pair of vertices $i, j \in V$ with an asymmetric cost $\lambda_{ij} \neq \lambda_{ji}$, the base graph is a directed graph $G = (V, A)$, where the arc cost of the arc set $A = \{(i, j) \in V \times V: i \neq j\}$ is λ_{ij}. The difference between the calculation of edges E and arcs A is: $|E| = n(n + 1)/2$ and $|A| = n(n + 1)$. In summary, the CVRP instance can be uniquely determined by a fully weighted graph $G = (V, E, \lambda_{ij}, q_i)$ or a directed graph $G = (V, A, \lambda_{ij}, q_i)$, as well as the number of vehicles $|K|$ and vehicle loading capacity Q. For the node-set $S \subseteq V$ that has been visited, for the undirected graph, the cut set (the set of edges of the connected graph G, such that removing these edges will separate G into two parts, but if one of the edges is removed, the graph will still be connected) is $\delta(S) = \{i, j \in E: i \in S, j \notin S\}$, while for directed graphs $G = (V, A)$, the in and out edges are defined as $\delta^-(S) = \{(i, j) \in A: i \notin S, j \in S\}$ and $\delta^+(S) = \{(i, j) \in A: i \in S, j \notin\}$. $A(S) = \{(i, j) \in A: i, j \in S\}$ is a collection of all arcs connecting vertices in S.

A complete vehicle route (tour) can be expressed as $r = (i_0, i_1, \dots, i_s, i_{s+1})$, where $i_0 = i_{s+1} = 0$ and the customer collection $S = \{i_1, \dots, i_s\} \subseteq N$ is visited. The cost of the route r is $\lambda(r) = \sum_{p=0}^{s} \lambda_{i_p, i_{p+1}}$. If the capacity constraint $q(S) = \sum_{i \in S} q_i \leq Q$ is established, and no customer is visited more than once, the vehicle route is feasible. That is, for all $1 \leq \beta < \gamma \leq s$, $i_\beta \neq j_\gamma$, $S \subseteq N$ can be said to be a feasible cluster or a feasible cluster at this time. A CVRP solution consists of $|K|$ feasible routes, where each vehicle $k \in K$ corresponds to a feasible route. If all routes are feasible and the clusters form a N partition, then the routes $r_1, r_2, \dots, r_{|K|}$ and the corresponding clusters $S_1, S_2, \dots, S_{|K|}$ are a viable solution provided by CVRP. CVRP consists of two interdependent tasks:

(1) Divide the customer collection N into feasible clusters $S_1, S_2, \dots, S_{|K|}$;

(2) Each vehicle $k \in K$ passes the route in $0 \cup S_k$.

Note that (2) is equivalent to the solution to the traveling salesman problem (TSP). The two tasks are mutually reinforcing because the cost of the cluster depends on the path, and the path requires the cluster as input.

2.2. A model of operations research

VRP (or TSP) contains many variants according to different application scenarios and realistic conditions [4,13,14], such as MTSP (multiple traveling salesman problem), VRPTW (vehicle routing problem with time window), FSVRP (multi-vehicle vehicle routing problem), VRPM (vehicle routing problem for multiple uses of vehicles), VRPSD (random demand vehicle routing problem), VFP (vehicle filling problem), VSP (vehicle scheduling problem), etc. Traditional algorithms usually design corresponding solutions for each problem. Still, we can apply deep RL to learn the problems' internal structure and a "meta-algorithm" to generalize them to different problems with similar structures. Next, we give a complete model description of the most basic CVRP.

The most basic CVRP model has integer decision variables x_{ij} for $\{i, j\} \in E$ (or $(i, j) \in A$), indicating that the vehicle moves directly between i and j. We present the most basic directed graph in the way of operations research, specifically:

$$\min \sum_{(i,j) \in A} \lambda_{ij} x_{ij} \tag{1}$$

$$s.t. \sum_{j \in \delta^+(i)} x_{ij} = 1, \forall i \in N \tag{2}$$

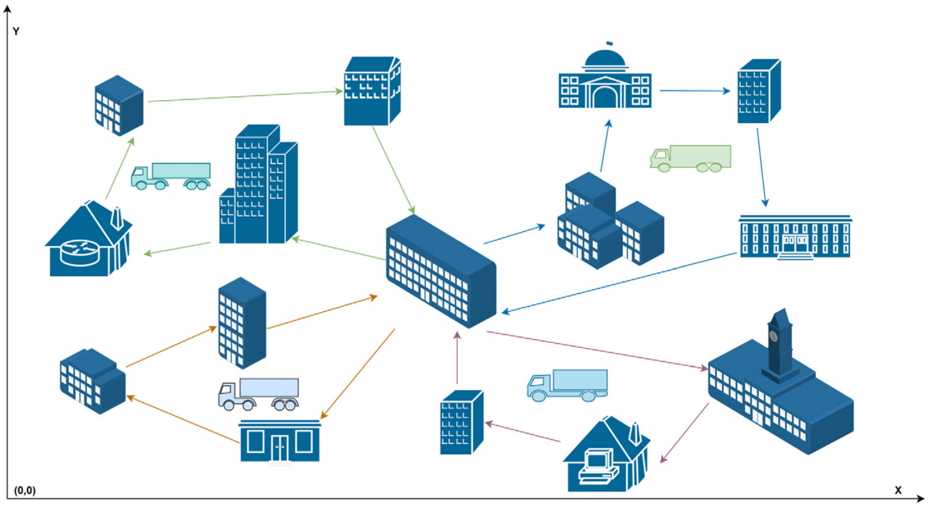

Fig. 1. A simple representation of a realistic transportation graph, in which the building at the center represents a central warehouse, and the remaining buildings represent scattered customers with different demands. We need to plan a set of routes so that the total cost of goods being distributed from the central warehouse to each customer node by vehicles with the same or different load capacities is the lowest. In the graph, we can usually know the coordinate position of each node to obtain the distance between each node, while the total cost is usually the shortest total distance. Similar problems, including the facility location problem (FLP), the three-dimensional bin packing problem (3D-BPP), and the vehicle scheduling problem (VSP), can be based on path optimization to minimize the objective function (generally cost).

$$\sum_{i\in\delta^-(j)} x_{ij} = 1, \forall j \in N \tag{3}$$

$$\sum_{j\in\delta^+(0)} x_{0j} = |K| \tag{4}$$

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} \geq r(S), \forall S \subseteq N, S \neq \varphi \tag{5}$$

$$x_{ij} \in \{0, 1\}, \forall (i,j) \in A \tag{6}$$

The objective function (1) is equivalent to minimizing the overall route planning cost. Constraints (2) and (3) indicate that each client node is connected to two other vertices in a path: its predecessor and successor. Constraint (4) represents the number of feasible routes from the warehouse to each customer node, which is the total number of vehicles $|K|$. Constraint (5) ensures the feasibility of the route. (6) means that the decision variable is an integer 0 or 1.

There are many types of combinational optimization problems in graphs. In addition to path-related problems, there are maximum vertex cover (MVC) problems, maximum cut (MC) problems, graph coloring problems, boolean operation problems, etc. Some of these problems can be converted to each other or have mathematical equivalence. We can apply machine learning to learn the internal structures of these problems to generalize to the same class of problems. There are various traffic problems derived from the VRP, such as location problems, vehicle scheduling, three-dimensional bin packing, etc., to optimize routes to minimize the total cost.

3. Combinatorial optimization with graph deep learning

3.1. An overview of deep learning

Machine learning is based on probability theory, statistics, and optimization [15] and is the foundation of big data, data science [16], prediction modeling [17], data mining, information retrieval, and other fields. Generally, we can divide machine learning into supervised learning (including semi-supervised learning), unsupervised learning, and reinforcement learning (RL) according to manually labeled data. Supervised learning derives predictive functions from artificially labeled data. Unsupervised learning derives conclusions and finds hidden structures from unlabeled data. RL learns how to select a series of actions from unlabeled data to maximize long-term benefits. The information available from training data in RL falls between supervised and unsupervised learning [15]. Supervised and unsupervised learning are typically one-time, short-sighted, and thinking about immediate rewards, whereas RL is sequential, far-sighted, and thinking about long-term cumulative rewards.

Deep learning [10,18], or deep neural networks, is a specific form of machine learning, usually used for supervised or unsupervised learning. It can be integrated with RL, often as a function approximator. Deep neural networks have one or more hidden layers between the input layer and output layer. They usually apply nonlinear transforms or activation functions (such as logistic, tanh, or Relu) on the unit's input to obtain an updated representation (Fig. 2). Then, in the output layer and each hidden layer, we can calculate the error derivative backwards and propagate the gradient back to the input layer to update the weight to optimize the loss function. Commonly-used and basic models in deep learning include the convolutional neural network (CNN) [19], the recursive neural network (RNN) [17] (including LSTM [20], GRU [21], etc.), and the residual network [22]. Normalization in neural networks includes batch normalization (BN) [23], group normalization (GN) [24], and ReZero [25]. Commonly used regularization techniques include dropout [26], early stopping [27], etc.

Feature engineering used to be done manually and was often time-consuming, over-specified, and incomplete. In deep learning, representation learning [28–30] can realize the end-to-end learning of automatic feature engineering and gradient descent, thus significantly reducing or even eliminating the dependence on domain knowledge. Distributed representation is the core idea of deep learning, which means that many features can represent each input, and each feature can represent many inputs. Distributed representations of depth can leverage hierarchies in the data to counter the curse of dimensionality. End-to-end training enables models to generate outputs using raw inputs that are not manually characterized, and the versatility, expressiveness, and flexibility of deep neural networks make some tasks easier. There are novel architectures and applications, such as micro-neural

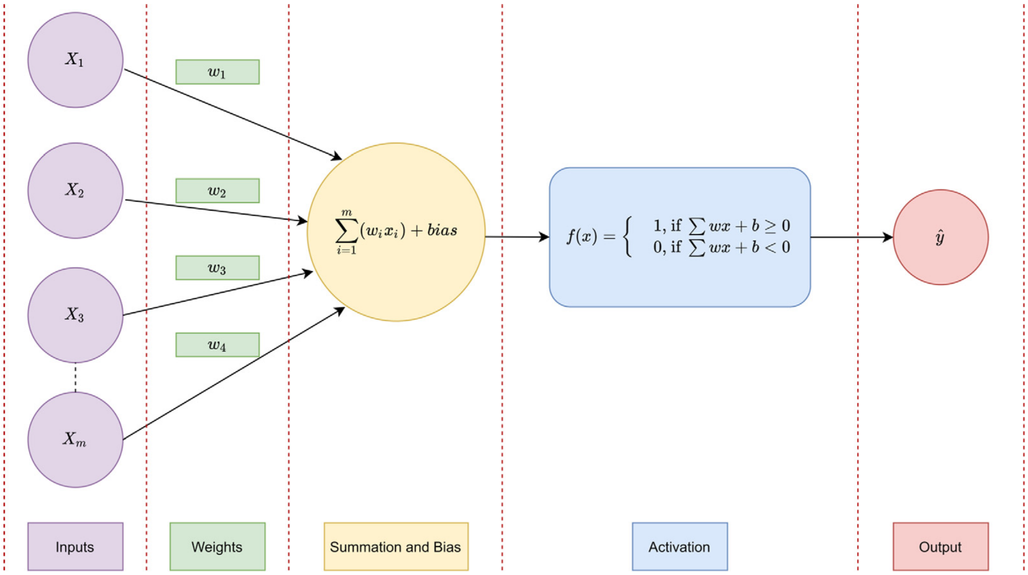
X_1
X_2
X_3
X_m
w_1
w_2
w_3
w_4
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1, \text{if } \sum wx + b \geq 0 \\ 0, \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
Inputs
Weights
Summation and Bias
Activation
Output

Fig. 2. The general flow of how artificial neurons work.

computers, asynchronous algorithms, question-answering systems, seq2seq for machine translation using original sentences, AlexNet for image classification using original pixels, DQN for games using original pixels and fractions, etc. [20,31–35].

3.2. Graph neural networks

A network or graph (collectively known as graphs in this section) is a data structure that models a group of objects (nodes) and their relations (edges) and is also common in data analysis problems. Graphs can be applied to social networks (social sciences) [36], physical systems (natural sciences) [37,38], knowledge graphs [39], bioinformatics [40,41], etc. Real-world graphs carry vital and rich information, and the information is not just from individual entities or parts [33]. Graphs are obtained as unique non-Euclidean data structures in machine learning and play an important role in downstream tasks such as search and recommendation. The general definitions of graphs (undirected graphs) and directed graphs are given below:

Definition 1 (*Graph*)**.** A graph can be given as $G = (V, E, A)$, where V represents a set of nodes, E represents a set of edges, and A is an adjacency matrix. $v_i \in V$ is used to represent a node, $e_{ij} = (v_i, v_j) \in E$ represents an edge, the adjacency matrix A is a $N \times N$ matrix, where if $e_{ij} \in E$, then $A_{ij} = w_{ij} > 0$, and if $e_{ij} \notin E$, then $A_{ij} = 0$. The degree of a node is the number of edges connected to it, and is formally defined as $\deg(v_i) = |N(v_i)|$, where $N(v_i)$ represents the set of all neighbors of the node v_i. When there is only one kind of node type and relationship type, it is called an isomorphic graph. On the contrary, when there is more than one kind of node type or relationship type in a graph, it is called a heterogeneous graph, and most graphs in the real world are heterogeneous graphs.

Definition 2 (*Directed Graph*)**.** Compared with an undirected graph, a graph containing directed edges is called a directed graph. For a directed graph, $A_{ij} \neq A_{ji}$, and in an undirected graph, $A_{ij} = A_{ji}$.

Definition 3 (*Attribute Graph*)**.** If the graph adds attribute information, it can also be called an attribute graph. The node attributes of the graph are expressed as X, where $X \in R^{N \times D}$ is a feature matrix, where N indicates the number of nodes and $N = |V|$, D indicates the dimension of a node vector, and $X_i \in R^D$ represents the feature vector of node v_i (in the case of $D = 1$, we employ X instead of $x \in R^N$ to represent the feature vector of the graph). Space–time is an attribute graph, which can be defined as $G = (V, E, A, X)$, where $X \in R^{T \times N \times D}$, and T are time steps.

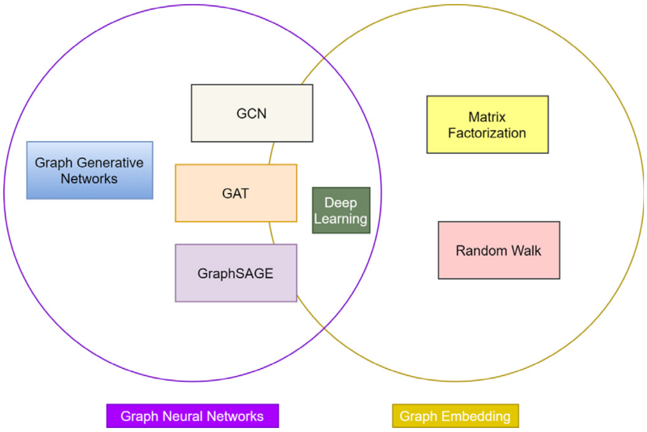

Fig. 3. The relationship between graph neural networks, graph embedding, and deep learning. In the figure, we list some representative GNN models, such as GGN (Graph Generative Networks) [42], GCN (Graph Convolutional Networks) [43], GAT (Graph Attention Networks) [44], GraphSAGE (Graph Sample Aggregate) [33], and graph embedding models, such as matrix factorization and random walk. GCN, GAT, and GraphSAGE have intersections with deep learning and graph embedding, while the intersection of GNN and graph embedding is deep learning.

3.3. Graph convolutional networks and graph embedding

The graph convolutional network (GCN) [43] generalizes the convolution operation of traditional data (images or grids) to graph data. It attempts to replicate the success of CNNs in image data by using graph theory or spatial locality to define graph convolution. The graph neural network (GNN) was proposed due to two essential motivations [37,42,45] (Fig. 3 demonstrates the internal relationship between GNNs, deep learning, and graph embedding).

One is from a convolutional neural network (CNN) [46]. CNNs can extract multi-scale local spatial features and combine them into highly expressive representations, thus making breakthroughs in almost all machine learning fields. Nonetheless, CNNs usually process regular Euclidean data such as images (2D grids) and text (1D sequences). The critical characteristics of CNNs are local connections, shared weights, and the use of multiple layers. Since graph data is the most typical local connection structure, shared weights lessen the amount of calculation compared with traditional spectral theory. The multi-layer structure is the key to hierarchical processing, to capture features of various sizes. Hence we can directly consider the promotion and application of CNNs to graphs. Yet standard neural networks such as CNNs and RNNs do not handle graph data input well because they superimpose features of nodes in a specific order, while there is no natural order of nodes in a graph. To solve this problem, as a representative algorithm of deep learning and graph data combination, GCN is propagated on each node separately, ignoring the input order of nodes. The output of a GCN is invariant under the input order of nodes. In a standard neural network, edges in a graph (information-dependent) are only a feature of nodes, and a GCN can propagate messages under the guidance of a graphical structure rather than as part of a feature. In general, GCNs update the hidden state of a node by the weighted sum of the node's neighborhood states. The essential technique is to learn a function and generate the representation of the node v_i by aggregating the features of the node X and the neighborhood representation v_i.

Another motivation comes from graph embedding (representation learning) [33,47–49], which aims to represent network vertices in a low-dimensional vector space by retaining network topology and node content information, thus completing any subsequent graph analysis task. The research of graph neural networks is closely related to graph embedding, which is also a topic of increasing concern in data mining and machine learning circles. Many graph embedding algorithms are typically unsupervised. We can roughly divide them into three categories: matrix factorization, random walk, and deep learning algorithms. Graph embeddings based on deep learning belong to graph neural networks. Inspired by word embeddings [50], DeepWalk [51] is the first graph embedding method based on representation learning. It takes a node as a word, a node sequence generated by random walk on the graph as a sentence, and processes it by the natural language model SkipGram [52]. Analogous methods such as node2vec [50], LINE [53], and SDNE [54] also achieved desirable results, whereas for large graphs, these methods may be computationally expensive and not optimal. With graph structure and node content information as input, the output of a GCN can be utilized on different graph analysis tasks through one of the following mechanisms [42,45]:

- Node-level output, such as tasks related to node regression and classification. Since GCNs directly give the potential representation, the last layer of GCN is generally a multi-level perceptron layer (MLP) or *softmax* layer.
- Edge-level output, such as edge classification and link prediction-related tasks. The function extracts the latent representation of the two nodes of the edge from the graph convolution model as input to predict the label/connection of the edge.
- Graph-level output and graph classification-related tasks. The pooling layer is used to coarsen the graph into subgraphs to "sum" or "average" the node representation to obtain a compact representation at the graph level.

3.4. Graph neural networks and their variant frameworks

If the proposal of graph convolutional networks (GCNs) [43] drives a broad class of approaches that apply deep learning (neural networks) to the learning task of graph-structured data, then graph neural networks (GNN) give a broader definition covering this class of approach. A GCN is essentially a process of iteratively aggregating node neighborhoods. Inspired by this, many approaches for improving and redesigning such aggregation operations have been designed to improve the adaptability and generalization of graph data. Through the analysis and deconstruction of such approaches, the general expression form and unified paradigm, namely the general expression framework of GNNs [42,45], is abstracted from a more unified level.

GNNs utilize graph structure and node features to learn the node representations h_i or the representation vectors h_G of the entire graph. GNNs follow the neighborhood aggregation strategy and iteratively update the representation of the node by aggregating the neighborhood representation of the node. After the k-th aggregation iteration, the representation of a node captures the structural information in its k-hop network neighborhood. Formally, the first layer of GNN is (many other GNN variants can be expressed similarly [55]):

$$a_i^{(k)} = AGGREGATE^{(k)}(\{h_u^{(k-1)} \colon u \in N(v_i)\}) \tag{7}$$

$$h_i^{(k)} = COMBINE^{(k)}(h_i^{(k-1)}, a_i^{(k)}) \tag{8}$$

$h_i^{(k)}$ is the feature vector of the node v_i in the k-th iteration or layer. The initialization representation vector $h_i^{(0)} = X_i$, $AGGREGATE(k)$, and $COMBINE(k)$ in GNNs are abstract generalization functions, which can be replaced with functions $MAX(k)$, $MEAN(k)$, etc., and it is crucial to choose the specific function. In the architecture of some *AGGREGATE* functions that have been proposed, such as the pooled variant of GraphSAGE [33], *AGGREGATE* is expressed as

$$a_i^{(k)} = MAX(\{ReLU(W \cdot h_u^{(k-1)}), \forall u \in N(v_i)\}) \tag{9}$$

W is a parameter matrix, $\cdot$ is the dot product, and the *MAX* represents a *MAX* pool divided by the number of elements. In a graph convolutional network (GCN), the aggregation and merge steps using the *MEAN* pool are integrated as follows:

$$h_i^{(k)} = ReLU(W \cdot MEAN\{h_u^{(k-1)}, \forall u \in N(v_i) \cup \{v_i\}\}) \tag{10}$$

Convolution can learn valuable features of nodes to solve many node-centered tasks. Nevertheless, to handle graph-centric tasks, one needs to aggregate the node's information to form a graph-level representation, which is often referred to as a readout or graph coarsening operation. Thus, for node classification, the node representation $h_i^{(k)}$ of the final iteration is used for prediction, while for graph classification, the function is read out to aggregate the node characteristics of the final iteration step to obtain the representation h_G of the whole graph:

$$h_G = READOUT(\{h_i^{(k)} | v_i \in G\}) \tag{11}$$

The readout operation of the graph is sequentially invariant and should remain the same no matter the order of the nodes. If we change the index of the nodes and edges using a bijection function between two sets of vertices, the representation of the entire graph should not change. The readout can be a simple permutation invariant function, such as summation, or a more complex graph-level pooling function.

In GCNs, neighborhoods of nodes are combined with equal or predefined weights, but the influence between neighbors will be very different, which will make the model better than the predefined ones in training. The attention mechanism [56] has been successfully applied to many sequence-based tasks, such as machine translation [57,58], machine reading [59], etc. Graph attention networks (GAT) [44] modify the convolutions of GCNs and introduce attention to GCNs, which incorporate the attention mechanism into the propagation step. They pay attention to the neighbors of each node, follow a self-attention strategy to calculate the hidden state of each node, and propose a single graph attention layer (GAL) constructed by superimposing this layer to construct an arbitrary graph attention network. It is necessary to normalize the correlation calculated with all neighbors to assign weights. The layer calculates the coefficients in the attention mechanism of the node pair in the following manner:

$$\alpha_{ij}^{k} = softmax_j(e_{ij}^{k}) = \frac{\exp(e_{ij}^{k})}{\sum_{v_l \in N(v_l)} \exp(e_{il}^{k})} \tag{12}$$

The complete weight coefficient calculation formula is as follows:

$$\alpha_{ij}^{k} = \frac{\exp(LeakyReLU(a^T[Wh_i^k \| Wh_j^k]))}{\sum_{l \in N(v_i)} \exp(LeakyReLU(a^T[Wh_i^k \| Wh_l^k]))} \tag{13}$$

a^T is the weight parameter, *LeakyReLU* is the activation function, and W is the weight matrix applied to the shared linear transformation of each node. If the above weight coefficient is calculated, the new feature vector of the node v_i is as follows by the idea of the weighted sum of the attention mechanism:

$$h_i' = \sigma(\sum_{v_j} \alpha_{ij}^{k} Wh_j) \tag{14}$$

At the same time, the structure of a GAT also has the following characteristics: (1) by assigning arbitrary weights to neighbors, it can be applied to nodes with different degrees; (2) the calculation of node and neighbor pairs are parallel, so the operation is effective; and (3) it can be easily applied to inductive learning.

The message passing neural network (MPNN) [60] is a graph-based general framework for supervised learning. The MPNN framework integrates the typical characteristics of several popular graph-structured data models, such as spectral methods and non-spectral methods [61], portal-graph neural networks [62], and deep tensor neural networks [63] in the graph convolution. MPNNs contain two phases: the messaging phase and the readout phase. First, the messaging phase runs T time steps, defined by the message function $MESSAGE_t$ and the vertex update function $UPDATE_t$. The update process for hidden state h_i^t and message m_i^t is as follows:

$$m_i^t = \sum_{j \in N(v_i)} MESSAGE_t(h_i^{t-1}, h_j^{t-1}, e_{ij}) \tag{15}$$

$$h_i^t = UPDATE_t(h_i^{t-1}, m_i^t) \tag{16}$$

The readout function applies the readout function to calculate a feature vector for the entire graph (complete time step):

$$y = READOUT\{h_i^T \,|\, v_i \in G\} \tag{17}$$

As mentioned earlier, the readout function can be set differently, so we can modify the readout function so that MPNNs will have different functions.

In addition, GNNs also have some variants, such as GAE [64,65], GCPN [41], Graph-LSTM [21], GN [37], etc. Their basic ideas are integrating GCN-based mechanisms with other deep learning components to form a new model or a new variant with more generalization ability or specific ability based on the fusion of several variants.

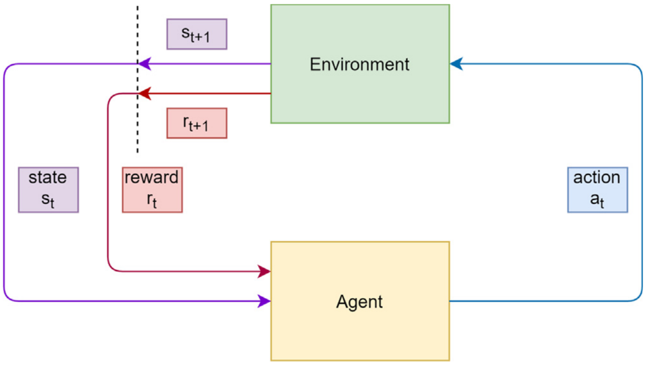

Fig. 4. The interaction process between agent and environment in RL.

3.5. Modeling combinatorial optimization problems with graph neural networks

In recent years, the application of deep learning to solve combinatorial optimization problems on graphs has been a research hotspot [66–68]. For example, in most combinatorial optimization problems, a decision sequence is involved, which is a sequence of decision problems. Hence, recursive neural networks (RNNs) and other methods in deep neural networks can ultimately realize the mapping from one sequence to another [69]. Pre-processing for graph data requires the representation ability of deep learning. It is feasible to exploit deep neural networks to solve combinatorial optimization problems. Some recent works have exploited graph neural networks to build network frameworks based on the characteristics and attributes of graph structures. This section briefly introduces several recent works adopting graph neural networks to solve combinatorial optimization problems on graphs. For a detailed introduction of related work, please refer to Section 5.

Khalil et al. use graph neural networks to learn heuristic algorithms on graphs [70]. They first employed structure2vec [71] (a variant of the GCN) to obtain node embeddings (low-dimensional representation of nodes) and then input it into Q-learning [34,72] to carry out policy network training. They have achieved better results than previous approaches, fully proving the representational ability of graph neural networks. Kool et al. proposed a model based on attention [73], where the encoder can be regarded as a graph attention network (GAT) [44], and its readout stage is an attention-based decoder. Experiments have proved the efficiency and effectiveness of the approach, and the graph neural network itself can show strong reasoning ability without reinforcement learning. At the same time, they employ the GAT to take the entire graph as input instead of the sequence as input. It is better to adopt the graph as input than the sequence as input because the nodes are arranged anyway. If the given graph is unchanged, then the output will not change, which is an advantage over the previous input sequence methods and the advantage of the graph neural network's ability to analyze and process the entire graph.

4. Deep reinforcement learning

RL is an area of machine learning that focuses on the idea of how a software object should behave in an environment to maximize cumulative rewards [74,75]. Through trial and error, the agent interacts with the environment to learn an optimal policy to solve sequential decision problems in various fields in the natural sciences, social sciences, and engineering (Fig. 4). The sequential decision process is as follows: the agent is provided with some initial observations of the environment and is required to select some operations from a given set of possible operations. The environment responds by switching to another state and generating a reward signal (scalar number), which is envisaged as a basic authenticity estimate of the agent's performance. The process continues, and the agent makes choices based on its observations and circumstances and reacts to the next state and reward signal. The sole goal of the agent is to maximize the cumulative reward [76]. Several key assumptions have been introduced in the description of this learning process model: first, because the interaction between the agent and the environment is continuous, the time-space is considered discrete; second, the environment provided contains some reward function—this is a manifestation of the reward hypothesis, also known as the RL hypothesis. RL is inherently used to solve sequential decision problems. We can utilize deep learning to deal with graph data in combinatorial optimization problems. Hence for sequential decision problems in combinatorial optimization, the direct utilization of RL is entirely feasible [66,77,78].

The process of a RL agent interacting with the environment over time can be described as a Markov decision process (MDP): at each time step t, the agent receives a state s_t in the state space S, follows a policy $\pi(a_t|s_t)$ to select action a_t from the action space A (where the policy is the behavior of the agent, that is, the mapping from state s_t to action a_t), receives a scalar reward r_t, and transitions to the next state s_{t+1}, for some reward function $R(s, a)$ and state transition probability $P(s_{t+1}|s_t, a_t)$. This process continues until the agent reaches its final state, and then it restarts. The total reward $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the cumulative reward after discount, and the discount factor is $\gamma \in (0, 1]$ [79].

When we employ deep neural networks to approximate any component of RL, we will develop deep RL [79], including components such as a value function $\hat{v}(s; \theta)$, policy $\pi(a|s; \theta)$, and model (state transfer function and reward function), where the parameter θ is the weights of the deep neural network. Deep RL combines the perceptive ability of deep learning and the reasoning ability of RL. Deep learning enables RL to expand to previously difficult decision-making problems, environments with high-dimensional states, and larger action spaces. In recent work in the deep RL field, two successful cases have attracted worldwide attention. First, the revolution of deep RL started with an approach that can learn to play a series of Atari 2600 video games at a superhuman level directly from the image pixels, which provides the instability of the function approximation technology in RL solutions [74]. It convincingly proves that the RL agent can be trained in primitive, high-dimensional observations, and is entirely based on reward signals. The second outstanding success was creating a hybrid deep RL system AlphaGo, which defeated a human world champion in Go [80]. AlphaGo is comprised of neural networks, which train the model by combining supervised learning and RL with traditional heuristic search algorithms.

Similarly, AlphaGo Zero [81], AlphaZero [75], MuZero [82], etc., are leading the revolution in artificial intelligence. Deep RL can be subdivided into value function-based learning and policy gradient-based learning. The following are some of the most representative or latest relevant algorithms.

4.1. Value-based algorithms

Value-based algorithms obtain the approximation of the optimal Q function $q(s, a)$ through dynamic programming and construct the optimal implicit policy [34]. In deep RL, a neural network is utilized to represent the Q function [83], and supervised learning is utilized for approximation dynamic programming. The value function is the prediction of cumulative, discounted, and future rewards and measures how good or bad each state or state–action is.

Solving RL tasks usually leads to a policy that not only maximizes the expected reward of the start state s_0, but of any state $s \in S$, which is derived from the Markov property: from a specific step t, the reward being collected does not depend on the previous history, and for the agent in a state s, performing the optimal task is equivalent to maximizing the expected reward with the current state s as the starting state. Many RL algorithms seek the optimal policy and additional information about the usefulness of each state. For a given MDP and policy π, the value function under the policy π is defined as:

$$V^{\pi}(s) = E_{\tau_\pi|s_0=s} \sum_{t=0} \gamma^t r_{t+1} \tag{18}$$

The value function estimates the performance of the agent using the policy π to access the state s and generalizes the concept of discounted expected reward $J(\pi)$ corresponding to $V^{\pi}(s_0)$. Since the value function can be derived from any policy, the value function $V^{\pi^*}(s)$ under the optimal policy π^* can also be considered, and it can also be expressed as $V^*(s)$ and is called the optimal value function.

4.1.1. Temporal-difference learning

Temporal-Difference (TD) learning [11] is one of the core technologies of RL. TD learning usually refers to the proposed learning methods for value function evaluation. SARSA and Q-learning are also regarded as TD learning. TD learning learns the value function $V(s)$ directly from incorrect experiences with bootstrapping, model-free, online, and fully incremental methods. TD learning is a prediction problem, and its update rule is $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$, where α is the learning rate and $r + \gamma V(s') - V(s)$ is called the TD error. Bootstrapping is the same as the TD update rule, which estimates the state or action value based on subsequent estimates, such as TD learning, Q-learning, and actor–critic algorithms. Bootstrapping methods usually learn faster and enable learning to be online and continuous.

The bootstrapping method is not an example of actual gradient descent (because the target depends on the weight to be estimated), but an example of semi-gradient descent.

SARSA stands for state, action, reward, (next) state, (next) action, and is a policy control method based on update rules:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \tag{19}$$

Q-learning is a non-policy control method to find the optimal policy and uses update rules to learn action-value functions:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \tag{20}$$

Q-learning greedily improves the action-value policy through the max action. TD learning, Q-learning, and SARSA converge under certain conditions. From an optimal action-value function, we can obtain an optimal policy.

Monte Carlo methods do not utilize dynamic programming to guide the value function. Still, they estimate the expected return value function of the state through the return of multiple rollouts of the averaging policy [11]. Thus, pure Monte Carlo methods can also be adopted in non-Markovian environments. On the other hand, they can only be used in limited MDPs because the rollout must be terminated to calculate the reward. As the $TD(\lambda)$ algorithm combines TD learning and Monte Carlo policy evaluation, the best results of these two methods can be obtained. A discount factor λ in $TD(\lambda)$ is used for interpolation between Monte Carlo evaluation and bootstrapping.

4.1.2. Deep Q-learning (DQN)

Modeling a policy or Q function using a neural network frees one from constructing task-specific features and opens up the possibility of applying RL algorithms to complex tasks (for example, tasks with an image as input). A video game is a classic example of such a task, where the raw pixels of the screen are provided as a state, accordingly, as input to a policy or Q function. The main idea of deep Q-learning is to adopt the TD algorithm, and its updating formula is as follows:

$$Q_{t+1}^{*}(s,a) = \begin{cases} Q_t^{*}(s,a) + \alpha_t[r_{t+1} + \gamma max_{a'} Q_t^{*}(s_{t+1},a') - Q_t^{*}(s,a)], \\ \quad if\ (s = s_t, a = a_t) \\ Q_t^{*}(s,a),\ else \end{cases} \tag{21}$$

It is equivalent to gradient descent and trains neural networks to solve a regression task. DQN [84] has made some significant contributions: (1) using experience replay and the target network, using deep neural networks (CNN) to stabilize the action-value function approximation training; (2) using only pixels and game scores as input, designing end-to-end RL methods so that only minimal domain knowledge is required; (3) the same algorithm, network structure, and hyperparameters can be utilized to train flexible networks to perform well on many different tasks, such as Atari games [74]. There are many extended variants of DQN, such as double DQN [85], dueling DQN [86], and value-based distributed DQN algorithms, such as QR-DQN [87], rainbow DQN [88], etc.

4.2. Policy gradient algorithms

Policies map states to actions, and policy optimization aims to find an optimal mapping. As described by Peters et al. [89], the range from direct strategy search to value-based RL includes: evolution strategy, CMA-ES (covariance matrix adaptation–evolution strategy), staged REP (relative entropy policy search), policy gradient, PILCO (Probability Inference for Learning Control) [90], model-based REPS, policy search through trajectory optimization, actor–critic algorithms, natural actor–critic algorithms, dominant weighted regression, conservative policy iteration, Q-learning and fitted Q, and extended variants such as contextual policy search and hierarchical policy search.

Gradients can provide a powerful learning signal for improving parameterized policies. In the more common model-free RL setting, the Monte Carlo estimate of the expected return is determined. At the same time, for gradient-based learning, the gradient cannot be applied to a sample of these random functions, which is a challenge for the Monte Carlo approximation. REINFORCE [91] is a representative policy gradient-based algorithm that provides a Monte Carlo-based estimation method to approximate the gradient (for any MDP and differentiable policy π_θ):

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta} \sum_{t=0} \nabla_\theta \log \pi_\theta(a_t\,|s_t\,)Q^{\pi}(s_t, a_t) \tag{22}$$

The Q function under the policy π is approximated by the corresponding regression:

$$Q^{\pi}(s,a) = E_{\tau \sim \pi_\theta |s,a} R(\tau) \approx R(\tau),\ \tau \sim \pi_\theta\,|s,a \tag{23}$$

The estimation is unbiased because the approximation of Q^π and the expected approximation on the trajectory are both done using the Monte Carlo method. Since the gradient estimation is unbiased, stochastic gradient ascent or more advanced stochastic optimization techniques are known to converge to a local optimum.

4.2.1. Actor–Critic Algorithms

Actor–critic algorithms learn both the policy and the value function used for guidance, updating the state from subsequent estimates to minimize variance and speed up learning [92,93]. The actor (policy) learns by using the feedback from the critic (value function). In doing so, these approaches balance the variance reduction of the policy gradient against the bias introduced by the value function approach. The only fundamental difference between actor–critic methods and other baselines is that actor–critic methods use value functions as baselines for the policy gradient. Instead of using the average value of the Monte Carlo reward as the baseline for the policy gradient approach, the actor–critic approach combines the advantages of policy search with value functions that can be learned from total gains and TD errors. They can benefit from both improved policy gradient approaches and value function approaches [94].

4.2.2. Deep Deterministic Policy Gradient (DDPG)

Policies are usually random, but Lillicrap et al. proposed a deterministic policy gradient (DPG) to estimate the policy gradient [77] effectively. Silver et al. introduced the DPG algorithm for continuous action space RL problems [95]. The deterministic policy gradient is the expected gradient of the action-value function in the state space. In contrast, the policy gradient is the desired gradient in the state space and the action space in the random case. The deterministic policy gradient can be calculated more efficiently than the random policy gradient estimation. Lillicrap et al. introduce a non-policy actor–critic algorithm to learn the deterministic target policy from the exploratory behavior policy. They apply the consistent function approximation of the deterministic function gradient to ensure an unbiased policy gradient. By extending DQN and DPG, Lillicrap et al. proposed a model-free, deep deterministic policy gradient (DDPG) algorithm in a continuous action space [93]. Actor–critic methods in the DDPG algorithm avoid action optimization at each step, making it have the same greedy strategy as Q-learning. But they also make it infeasible to be an approximator in the face of complex action spaces with huge unconstrained functions (such as deep neural networks). The DDPG algorithm adopts the experience replay similar to DQN and the idea similar to the target network to make learning more stable and robust, namely using a "soft" target. Instead of directly copying weights like DQN, it slowly updates soft target network weights θ' to track and learn network weights $\theta\!: \theta' \leftarrow \tau\theta + (1-\tau)\theta'$. As a non-policy algorithm, the DDPG algorithm learns from the experience of exploring the policy by adding noise to the actor policy. The DDPG algorithm can solve a problem with 20 times fewer empirical steps than DQN, although it still requires a lot of training to find a solution, just like most model-free RL methods [11].

5. Combinatorial optimization with deep reinforcement learning

5.1. Classic algorithms

The VRP and TSP (and related routing problems) are widely studied combinatorial optimization problems [12]. For Euclidean and non-Euclidean graphs, many approximation algorithms have been proposed.

Exact algorithms mainly apply linear programming, nonlinear programming, integer programming, and other mathematical programming techniques to describe the quantitative relationships in the system and find the optimal policy. They are usually applied to solve for the precise structure, clear boundary, and transparent relationship between elements. Nonetheless, due to the computational complexity (the complexity of the most well-known TSP exact dynamic programming algorithm is $O(2^n n^2)$, which increases exponentially with an increase in the scale of the problem (the number of nodes and edges)) of these algorithms, they cannot scale to larger combinatorial optimization problems. Moreover, in practical application scenarios, the structure, boundary, and elements (variables) are unclear. Even exact algorithms can be used to solve a problem specifically, they do not have generalization ability. That is, the solution to one problem does not apply to other variants.

An approximation algorithm is a type of progressive algorithm to solve combinatorial optimization problems. They can approximate the optimal solution by polynomial-time algorithms. An approximation algorithm can be used to solve combinatorial optimization problems with a large data size (relative to the exact algorithm). Still, approximation algorithms often produce poor results in the worst case of practical problems (the approximation ratio between the approximate solution obtained by approximation algorithms and the actual optimal solution is no more than 2, within two times of the total optimal cost). Moreover, some combinatorial optimization problems in practice cannot be approximately solved.

Heuristic search [96] refers to fast and practical algorithms for solving combinatorial optimization problems that lack theoretical support, such as genetic algorithms, ant colony algorithms, simulated annealing algorithms, etc. Nevertheless, the design of existing heuristic algorithms relies on expert experience, professional knowledge, and trial and error. Some solution solvers have fused heuristics to solve large-scale problems. The most accurate TSP solver, Concorde, describes how to guide the space of feasible solutions in an effective way to solve symmetric TSP instances with thousands of nodes through a carefully hand-designed heuristic. It applies the cutting plane algorithm [97] to iteratively solve the TSP's linear programming relaxation and combines with the branch and bound algorithm to trim the part of the search space that does not contain the optimal solution. Similarly, the Lin–Kernighan–Helsgaun algorithm is inspired by the Lin–Kernighan heuristic and designs of symmetric TSP approximation heuristic search algorithms [97], and has been proved to be the optimal solution in instances with hundreds of nodes.

More general solutions, such as Google's OR-tools, combine local search algorithms and meta-heuristics to solve a superset of TSP problems. Local search algorithms apply a specific set of local mobility operators to candidate solutions based on artificial heuristic algorithms such as 2-opt [98], and navigate from one solution to another in the search space. The meta-heuristic algorithm is utilized to propose the uphill motion and avoid the optimal local solution [99]. A popular option for the meta-heuristic approach of the TSP and its variants is to guide local searches, which avoid local minima by penalizing specific solution characteristics that it believes should not appear in a desirable solution.

Applying existing heuristic search algorithms to newly encountered problems or new instances of similar problems is challenging because all search algorithms have the same performance when averaging all problems. Hence, the search algorithm must be selected appropriately based on the priority of the problem to ensure performance. This challenge is the underlying motivation behind hyper-heuristics, defined as "the search algorithm or learning mechanism that selects or generates heuristics to solve computational search problems". Hyper-heuristics [100] are knowledge-intensive processes that partially abstract heuristics for a given combinatorial problem, are easier to use than problem-specific algorithms, and have been demonstrated to successfully combine human-defined heuristics in a superior way in various tasks. Nevertheless, hyper-heuristic algorithms run on the search space of the heuristic algorithms rather than on the search space of the solutions, and hence still initially rely on the heuristic algorithms generated by humans.

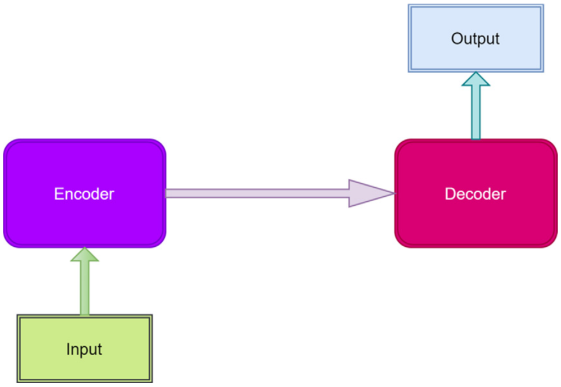

Fig. 5. Based on the general frame structure of a sequence-to-sequence model, the encoder encodes or embeds the input, and the decoder decodes the processed data into output.

5.2. *Related neural network architectures*

The application of neural networks in combinatorial optimization problems can be traced to Hopfield&Tank [67,101], who proposed the Hopfield network to solve small TSP problems. The Hopfield network is a type of recursive neural network, which combines storage systems and binary systems. It guarantees convergence to a local minimum, but there is likewise a probability of convergence to a non-global minimum or an incorrect local minimum [102]. In recent years, deep learning models have been increasingly used to address combinatorial optimization problems.

5.2.1. *Sequence-to-sequence models*

Although neural networks are flexible, they can only be adapted to problems where the input and target can be reasonably encoded with vectors of fixed dimensions. This is a key constraint, and many crucial problems are better represented by sequences of unknown length, rather than requiring that the input and output dimensions are known and fixed. For example, a question answering system can be thought of as mapping a set of words representing a question to a set of words representing an answer. Speech recognition and machine translation are both sequential problems. So, it makes sense to learn a domain-independent way of mapping sequences to sequences.

Sutskever et al. proposed a general end-to-end sequence learning method (Sequence-to-Sequence or seq2seq). The general structure including the encoder and decoder is illustrated in Fig. 5 [20]. It applies a multi-level LSTM to map the input sequence to a fixed dimension vector and then applies another deep LSTM to decode the target sequence (as illustrated in Fig. 6). They demonstrate that an LSTM architecture can solve general sequence-to-sequence problems. The core idea is to apply an LSTM to read the input sequence a time step at a time, obtain the fixed dimension vector, and then adopt another LSTM to output the sequence extracted from the embedding. The two LSTMs are the encoder and decoder, and the second LSTM is essentially an input sequence for the recursive neural network model of language [103].

LSTMs or other variants of RNNs, such as GRUs, were chosen because of their success in learning data with long time

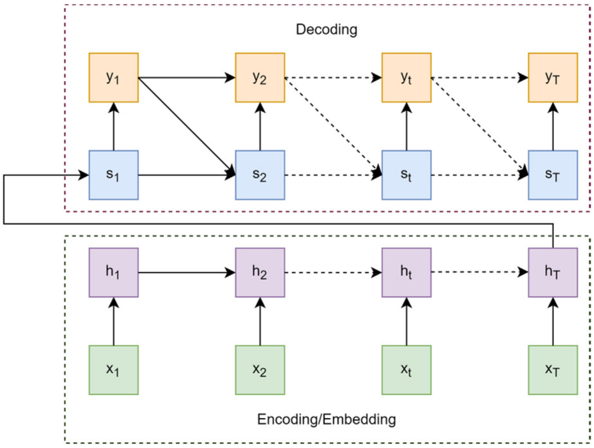

Fig. 6. Sequence-to-sequence framework. x is the input, h is the LSTM encoded vector, s is the state sequence, and y is the output decoded from the state sequence.

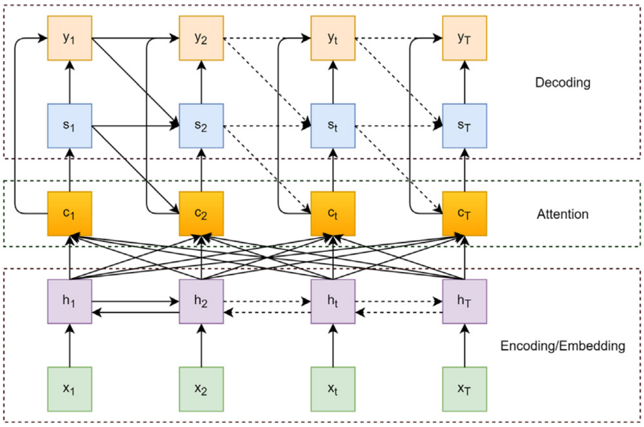

Fig. 7. Seq2Seq model with an attention mechanism.

dependencies, with considerable time delays between the inputs and their corresponding outputs. The seq2seq model encodes the source sequence as a vector of fixed dimension, which will lose a lot of information for long sequences. Bahdanau et al. introduced an attention mechanism based on seq2seq and proposed to store the layer state h_i in a list, and then when decoding next, the T hidden layer states and their correlation are calculated according to the state s_{t-1} at the previous moment, and the weighted sum is used to obtain the encoding information c_i, that the c vector at each decoding moment is different. The weight adjustment and calculation were performed according to the correlation between output and input (see Fig. 7) [57].

5.2.2. Pointer networks

Vinyals et al. introduced a neural network (Pointer Network) based on a Sequence-to-Sequence network to learn the conditional probability of the output sequence. The elements are discrete labels corresponding to the positions in the input sequence [104]. In a Sequence-to-Sequence model, the number of target classes in each step of the output depends on the input length, and the length of the input is variable. Usually, the content of the output sequence of a combinatorial optimization problem is the same as the content of the input sequence. Still, the order of the elements in the sequence has changed. In many cases, a combinatorial optimization problem is to make a sequence decision. The pointer network is a very targeted neural network architecture for solving combinatorial optimization problems.

The pointer network [104] uses an attention mechanism [56] to solve the problem of variable-size output dictionaries (each element of the output sequence has some connection with one or more elements of the input sequence). It does not adopt attention in each decoding step to mix the hidden units of the encoder into the context vector. Still, it uses attention as a pointer to select the members of the input sequence as output. Specifically, it obtains the output sequence by the weight of an element of a position associated with each position of the input sequence. Then the input sequence and the weight are combined in a certain way to affect the output. In this process, each input element can only be pointed to by one output element to avoid the repeated occurrence of input elements.

5.2.3. Transformers

Sequential transformation models such as seq2seq and pointer networks are based on recursive or convolutional neural networks that usually perform factor calculations using the symbol positions of the input and output sequences, align the positions with calculation time steps, and then generate a sequence of hidden states h_t as a function of the previous hidden state h_{t-1} and position t. The inherent order precludes parallelization in training examples. Still, parallelization becomes critical when sequence lengths become long because memory limits batch processing. Some works have achieved significant improvements in computational efficiency through factorization techniques [66] and conditional calculations [105] while improving model performance, but the fundamental constraints of sequential calculations still exist.

Attention mechanisms have become an essential part of various sequence modeling and transformation models because they allow the modeling of dependencies, regardless of their distance in the input or output sequence [57]. Vaswani et al. proposed a simple network structure, the transformer framework [56], which does not require recursion or convolution but is based entirely on an attention mechanism to describe the global dependency between input and output. It allows more significant parallelization in translation experiments. There are more improvements based on the transformer, which are also used in areas such as image data, audio data, graph data, etc. [106–108]. The transformer also has an encoder–decoder structure, in which a multi-head attention mechanism (or self-attention mechanism) can be used in three different ways:

(1) In the "encoder–decoder-attention" layer, the queries come from the previous decoder layer. The memory's key and value come from the encoder's output, which allows each position in the decoder to participate in all positions in the input sequence.
(2) The encoder contains self-attention. In the self-attention mechanism layer, all keys, values, and queries come from the same position, and each position in the encoder can handle all positions.
(3) The self-attention layer allows each position in the decoder to contain all the position information. To maintain the autoregressive property of the decoder, the leftward information flow in the decoder needs to be prevented.

5.3. Review of research status

In recent years, while moving ahead with artificial intelligence and hardware computing capabilities, academia and industry have increasingly paid attention to applying machine learning methods to solve traditional NP problems [66]. Deep learning's perception ability and reinforcement learning's reasoning ability can be applied to combinatorial optimization problems to process large-scale data. Its robustness and generalization will be more superior to traditional methods. The two essential motivations for adopting deep reinforcement learning are to approximate and discover new policies [77]. How deep RL participates in solving combinatorial optimization problems is illustrated in Fig. 8.

In some works, researchers assume that they have a theoretical basis or empirical knowledge of combinational optimization algorithm decisions but hope to alleviate the computational burden by using deep learning to approximate some of these decisions. The motivation for using RL is that sometimes the expert knowledge of traditional algorithms is not satisfactory. Researchers hope to find better decision-making methods, and deep RL can train models through trial and error. The deep neural networks and Markov decision processes introduced in the previous chapters theoretically describe these two motivations. The environment is the algorithm's internal state, learning policies through experiential RL to discover new policies. In a simulation setting (traditional method), the policy is learned (and not rewarded) by the supervised object provided by the expert for each operation [109].

In contrast, in an empirical setting, the policy is learned from the use of RL (and no expert, possibly delayed) reward signals. The main difference between simulated settings and RL is that the agent is taught what to do, and the agent is encouraged to accumulate rewards quickly.

Recently, many researchers have reviewed machine learning and graph neural networks in combinatorial optimization (CO). But most of them are usually either summarized from a very general perspective or only focus on one aspect. For example, Bengio et al. [77] gave a high-level overview of CO's machine learning methods, which described the inherent relationship between machine learning and CO. They provided some essential theoretical basis and motivation for applying machine learning to CO. However, they focused more on learning in CO rather than modeling. Similarly, Kotary et al. [110] classified various machine learning methods in CO, mainly focusing on classifying end-to-end learning paradigms. Still, they made representation learning, especially graph neural networks, a secondary topic. Bai et al. [111] comprehensively reviewed a hybrid approach that combines analysis techniques with machine learning tools to solve VRP problems. They listed many state-of-the-art machine learning methods for VRP, but they did not systematically introduce the related technologies used in these methods. Mazyavkina et al. [112] focused on applying RL in CO (especially routing problems). They summarized some commonly used policy-based and value-based RL algorithms. Lamb et al. [113] and Cappart et al. [114] gave a high-level overview of the application of GNNs in various reasoning tasks. Still, they did not describe the specific role and process of learning in CO reasoning.

Compared with previous reviews, our review is more systematic, specific, and comprehensive. We systematically describe modeling, learning, and search in CO and describe the techniques, principles, and theories used in detail. More importantly, we systematically summarize the experimental methods, results, and analysis based on learning methods, which have rarely been seen in previous reviews. Finally, we summarize the challenges and future directions of learning-based methods based on a systematic analysis of almost all current research conditions introduced.

Next, we summarize in detail some state-of-the-art works based on deep RL in recent years. The current learning-based methods can be roughly divided into two categories: construction heuristics-based and improvement heuristics-based. The former uses the learned heuristic policies to construct solutions one by one from scratch, while the latter continuously improves the quality of the solutions by iteratively improving a given initial solution.

Bello et al. pioneered the utilization of RL (actor–critic framework) based on pointer networks to solve combinatorial optimization problems [115]. It is stated that although the heuristic method works well on TSP, once the problem statement differs slightly, the algorithm has to be modified. In contrast, machine learning may automatically apply its heuristics based on training data, which may suit various optimization tasks, requiring less manual engineering than a solver optimized for only one task. Nevertheless, if one adopts supervised learning when learning the mapping from training input to output, one cannot access the optimal label since the label is difficult to obtain in the combinatorial optimization instance.

Nonetheless, we can adopt a validator to compare the quality of a set of solutions and provide feedback for the learning algorithm. Therefore, we can follow the RL paradigm to cope with combinatorial optimization problems. Empirically, even using optimal solutions as labels, a supervised mapping has poor generalization ability compared to RL, that explores different paths and observes their corresponding rewards.

Two methods based on policy gradients are considered in their paper. The first method, called RL pre-training, applied a training set to optimize a recurrent neural network (RNN), which targets the expected return and parameterizes the random policy. At test time, the policy is fixed, and inference is executed by greedy decoding or sampling. The second method, called active search, did not involve pre-training. It started with a random policy, iteratively optimized RNN parameters on a single test instance, used the expected reward target, and tracked the sampling during the search process. They found that a combination of RL pre-training and active search works best in practice. On a 2D Euclidean graph with up to 100 nodes, neural combinatorial optimization is significantly better than the supervised learning method of TSP, allowing more calculation time while obtaining near-optimal results. They tested the same method on the backpack problem to prove the flexibility of their method and obtained the optimal solution for up to 200 project examples. These results reveal how neural networks can be used as general tools for solving combinatorial optimization problems, especially those that are challenging to design heuristic algorithms for.

Nazari et al. proposed an end-to-end framework [116] that uses RL to solve vehicle routing problems based on pointer networks and PN-AC [115]. The VRP is also defined as a Markov decision process (MDP), where the optimal solution can be viewed as a series of decisions. RL can be applied to generate an approximate optimal solution by increasing the probability of decoding the "expected" sequence, sampling from MDP trajectories to generate near-optimal solutions. The policy learned by some previous learning methods does not apply to other examples than those used in training. If the problem settings are slightly modified, the policy needs to be retrained from scratch. Numerous parallel combinatorial optimization problems are only changes of variables and data, and their internal structure does not differ. Therefore, they proposed a structure that can handle any problem sampling from a given distribution rather than training a separate model for each problem instance. The framework can be applied if it approximates the generated distribution of the problem, is an extension of PN-AC [115], and has more generalization capabilities because the trained model can be regarded as a black box heuristic (or meta-algorithm), generating high-quality solutions within a reasonable time. The pointer network assumes that the system is stationary for some time when decoding the solution. Still, in actual problems, the demand varies over time, so this process should be dynamic. To this end, they propose a more straightforward alternative model, which contains a recurrent

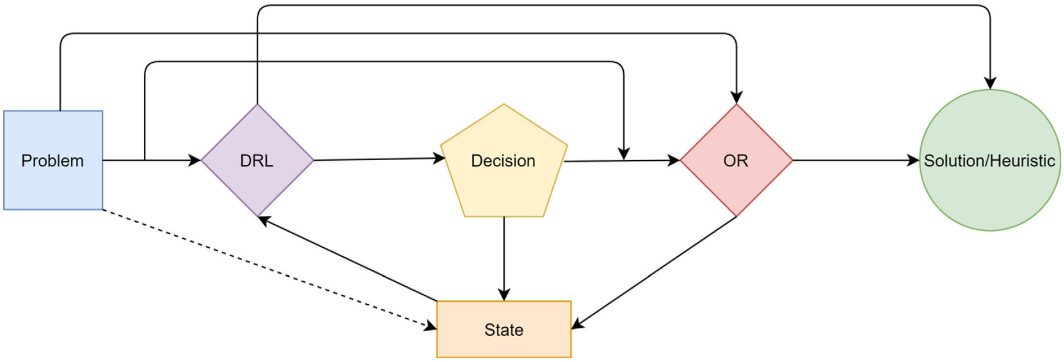

Fig. 8. There are three main ways in which deep RL (DRL) solves combinatorial optimization problems: (1) using the problem as input to give a solution or heuristic directly; (2) integrating or expanding traditional operations research algorithms (OR) to increase the value of valuable information. (3) traditional operations research algorithms iteratively querying the same deep RL model to make decisions. The DRL model takes the current state of the algorithm as input, which may include problems.

neural network (RNN) decoder and an attention mechanism. At each time step, an embedding of static elements is the input to the RNN decoder (using "embedding" instead of "encoding" in the pointer network). The output of the RNN and the dynamic element embedding is input into the attention mechanism. The attention forms a distribution on the feasible destinations that the next decision point can choose.

Khalil et al. exploited a combination of RL and graph embedding to solve combinatorial optimization problems on graphs [70]. The policy studied is similar to a meta-algorithm that incrementally constructs a solution whose operation is determined by a graph-embedding network on the current state of the solution. Based on the graph structure, they construct feasible solutions by constantly adding nodes and maintain the feasible solutions that satisfy the constraints of the graph. Greedy algorithms are generic patterns for heuristics for graph-related problems so that the same high-level design can be seamlessly applied to different graph optimization problems. Structure2vec is applied to represent the policy in the greedy algorithm. The new instance-graph-based deep learning architecture specializes the nodes in the graph, capturing the attributes of the nodes in their graph neighborhood context. It allows policies to differentiate nodes based on their usefulness and generalize them to problem instances of different sizes.

In contrast, previous sequence-based approaches did not take full advantage of the structure of the graph. In this paper, fitting Q-learning was applied to learn the greedy policy parameterized by the graph embedding network. The framework makes the policy's objective be to optimize the original problem instance's objective function directly. The paper also proved that the heuristic methods studied can maintain their effectiveness even when used on graphs much larger than those they were trained on.

Deudon et al. extended the neural combinatorial optimization framework and designed a data-driven heuristic algorithm to solve the travel agent problem (TAP) [12]. The model is based on the fact that many real-world problems generated by the OR community are solved daily from scratch using hand-crafted features and manually designed heuristics. They proposed a general framework for learning the heuristics of combinatorial tasks in which the outputs and inputs are in the same order. In the framework, the coordinates are used as input. The neural network in RL is used to enhance learning efficiency by designing a critic to calculate the baseline for a more extended patrol.

Furthermore, the authors strengthened their approach with the famous 2-OPT heuristic. The results demonstrate that the overall performance of the framework is comparable to that of high-performance heuristic algorithms (OR-Tools). When the framework is configured with a simple 2-OPT process, it can obtain near-optimal results on a two-dimensional Euclidean graph. It indicates that machine learning-based approaches can learn good heuristic algorithms and produce promising results compared to enhanced simple local search.

Kool et al. employed a deterministic greedy rollout as a baseline to guide the improved transformer to solve routing problems [73]. They used a greedy algorithm as the baseline, which is better than the traditional RL value function. Heuristics are used to predict the distribution of urban arrangements. Instead of using an RNN architecture, the proposed framework was expressed in rules that can be interpreted as policies for making decisions. Nevertheless, these policies can be parameterized using neural networks and trained to obtain new and more robust algorithms for many different routing problems. They aimed to design better models and better training methods to replace manual heuristic algorithms with learning routes. They suggested using a complete model based on attention and a simple but effective greedy algorithm as a baseline to reinforce the model's training. The baseline's goal is to estimate the instance's difficulty, and the instance's difficulty can be applied to estimate the algorithm's performance. The baseline is defined as the best model-defined policy for the cost of the deterministic solution produced by the greedy algorithm. The value of the proposed approach is not to exceed existing manual design heuristics for specific tasks but rather to extend to different path problems with a high degree of flexibility. It allows us to learn powerful heuristics for solving practical problems where desirable heuristics do not exist.

The attention model (AM) is an encoder–decoder model used to learn the input graph's representation and predict the correct path. Kool et al. modified the encoder to a GAT, where the input is the coordinates of each node, and the output is the nodes themselves. The graph is the average value of all the embedding nodes. The decoder also uses a GAT, whose input is associated with node embeddings, context or graph embeddings, the start node embedding, and the previous node embedding. The output is a series of nodes with the highest compatibility, selected at each step to add to the path. The accessed nodes are masked to ensure the feasibility of solutions.

Duan et al. proposed an approach founded on a GCN, which takes the node and edge feature embeddings as input [117]. They designed two independent decoders to decode the embeddings. One is a sequence prediction decoder based on a RNN, which takes node embeddings as input and outputs a sequence as the solution of the VRP instance. The other is a classification decoder based on a multilayer perceptron, which takes an edge embedding as input and outputs a probability matrix (which can also be transformed into the solution of a VRP instance). They used the output of the sequence prediction decoder as a supervised label for the output of the classifier decoder, which constitutes a mutually reinforcing mechanism while both two decoders are learning. They applied reinforcement learning to train the sequence prediction decoder and a supervised approach with policy sampling to train the classification decoder. The trained policy can be thought of as a black box heuristic (meta-algorithm) that can be used to solve a large-scale real-world VRP. Its performance is better than other existing learning models and is superior to the solution solver OR-Tools.

The above methods are all based on construction heuristics, which also include [36,69,118–120]. Methods based on construction heuristics can generate diversified solutions through the learned distribution, showing good generalization. However, the quality of the solutions is still slightly worse than that of a solution solver. At present, improvement heuristic-based methods that continuously improve the quality of the solution have surpassed solution solvers in solution quality. Next, we list several representative improvement-based methods.

The NeuRewriter algorithm [121] heuristically selects a part of the current solution through the learned policy to re-optimize and improve the performance until convergence. It divides the policy into two parts: region selection and rule selection, both of which are represented by a neural network and optimized simultaneously through reinforcement learning algorithms. It uses the advantage actor–critic algorithm and the learned Q function as critics to avoid boot-strapping, leading to insufficient samples and instability in training. The NeuRewriter algorithm has achieved good performance on expression simplification problems, job scheduling problems, and vehicle routing problems.

Lu et al. proposed a "Learn to Improve" (L2I) method [122], which is superior to operations research methods. The core idea is to add RL in the meta-heuristic iterative search to help select operators more effectively. Specifically, they first employed heuristics to construct the initial feasible solution, then employed RL to select an improvement operator to optimize the solution iteratively, and finally used a perturbation operator to avoid searching for the optimal local solution. In the process of exploiting an improvement operator, if it is determined that the current solution can be improved, the RL-based controller is applied to select an improvement operator and try to apply it to improve the solution. It tries to reduce the cost of the current solution by moving client nodes to different locations along each route.

On the other hand, when a local minimum is reached, the perturbation operator randomly destroys (in whole or in part) and reconstructs the perturbation operator of multiple routes to generate a new initial solution. Specifically, if there is no cost reduction for a limited number of improvement steps, we disturb the solution and start the improvement iteration again. Since perturbations can dramatically change the solution, the authors found it helpful to start over from a relatively desirable starting point for improvement iterations, filtering out restarted solutions that are significantly worse than the current solution or the current best solution.

Wu et al. proposed a framework [123] for directly learning the improvement heuristics. The framework improves the initial solution by iteratively performing a neighborhood search based on a particular local operator. Hand-made search policies guide traditional improvement heuristics, but these policies require much domain knowledge to design and may only bring limited improvements to the solution. Instead, they applied deep reinforcement learning to discover better policies for improvement automatically. Specifically, they first defined the improvement heuristic as a Markov decision process (MDP) in which the policy guides the choice of the next solution. Then, they proposed a self-attention-based policy architecture, through which they can combine a variety of commonly used pairwise local operators, such as 2-opt and node swapping. Finally, they applied the RL framework to TSP and CVRP and designed an actor–critic algorithm to train the policy network. On this basis, Costa et al. introduced a dual encoding mode, namely using GCN layers and sequence embedding layers, so that the pointing attention mechanism can be extended to more general k-opt moves [78].

Many other approaches are based on deep learning and RL, most of which first employ GNNs, attention, and RNNs to model combinatorial optimization problems and then use RL to train and optimize the entire framework [124,125]. Theoretically, a large class of similar problems can be solved using RL if the Markov decision process is altered according to the specific combinatorial optimization problem. Many combinatorial optimization problems are mathematically equivalent. That is, most of their internal structures are similar [70]. For example, besides VRP and TSP on graphs, RL can also solve MVC, vertex cover, max-cut, graph coloring, etc. [126].

AlphaGo [80] and AlphaGo Zero [81] represent the valuable features of Go and extract them through deep learning. Then, they combine Monte Carlo tree search and RL to self-play and update their network parameters, and have achieved great success in Go. They are also of immense significance for solving combinatorial optimization (CO) problems because there are similarities between Go and CO problems. For example, the search space of both is enormous, and if only exhaustive travel is expected to be adopted, the time consumed increases exponentially, which is unrealistic. AlphaGo uses a convolutional neural network and uses pieces and historical positions as features to build a value network and a policy network (AlphaGo Zero combines the two networks into one network). They adopt Monte Carlo tree search (MCTS) [128] to diminish the search width and depth.

MCTS is the core technology of the AlphaGo series, which applies the Monte Carlo method of random simulation to tree search [80]. Traditional search algorithms, such as minimax-search use an evaluation function to evaluate the value of a node (state) accurately. Still, when the search space is vast, they may exceed the hash rate. MCTS uses Monte Carlo simulations to estimate the node's value rather than an evaluation function [129]. MCTS can be divided into four stages: selection, expansion, simulation, and backpropagation, roughly described in Fig. 9. Traditional heuristic search algorithms commonly used for combinatorial optimization also include beam search, greedy algorithms, active search, meta-heuristic algorithms, local search, etc.

For combinatorial optimization problems, we can also establish appropriate mathematical models, exploit deep neural networks for appropriate feature representation, and combine appropriate search strategies to diminish the solution space. Similar to the win–lose rule of Go, a large number of combinatorial optimization problems, in reality, have a clear objective function and constraint conditions to evaluate the current policy, so that a model can be learned according to the reward and penalty signal of the expected result and that the model can maintain stable improvement in the training process.

We cannot obtain many optimal solutions to optimization problems as labeled data to train the network parameters. AlphaGo Zero has proven that deep learning and RL can perform complex tasks well without the guidance and assistance of human experience, even surpassing algorithms based on human experience. Algorithms based on expert experience tend to converge to a local optimum without knowing it. Still, machine learning can break through the limit. In combinatorial optimization problems, we can also gradually raise the model performance by self-playing with data and labels generated. At present, some

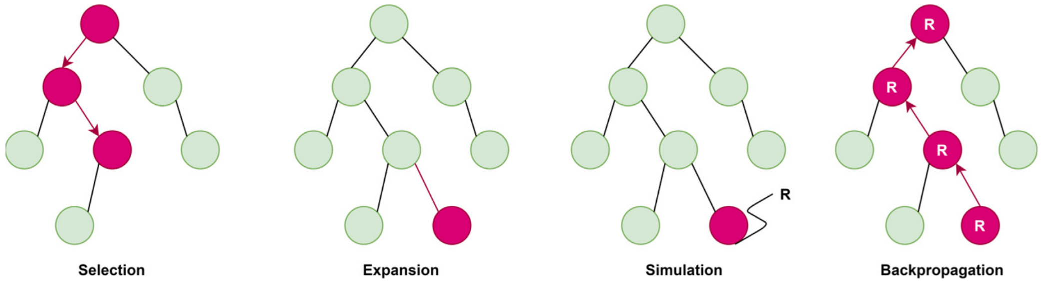

Fig. 9. The process of MCTS algorithms. Viewed from left to right, a path is first selected to maximize the UCB [127], second, the tree is expanded, third, the neural network performs the simulation, and finally, the results are propagated back to the update node along the current path.

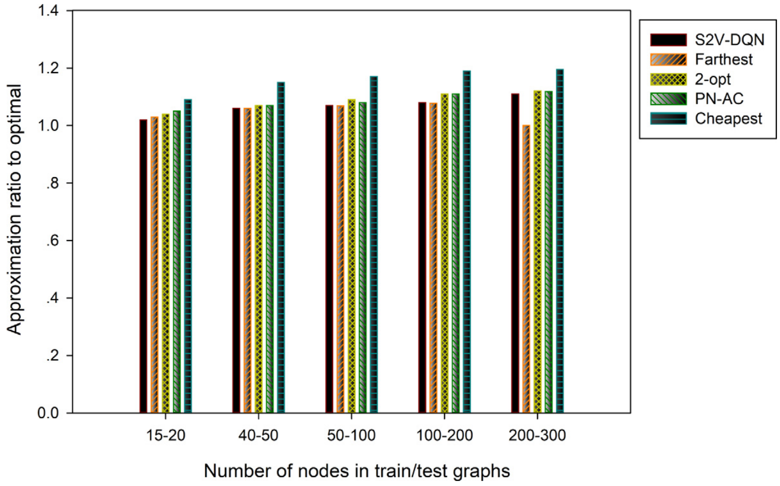

Fig. 10. The approximation ratio of the results obtained by different algorithms to the optimal solution on random TSP instances.

researchers have tried to exploit the ideas of the AlphaGo series for combinatorial optimization [35,75,82,127,130–134].

As a complete and general summary, we summarize the methods of using deep learning and RL for combinatorial optimization in recent years in Table 1, including the problems they solve, the deep learning models they employ, and the RL algorithms and their types. It is worth noting that MVC, MIS, MC, SAT, and MAX CUT can often be transformed into facility location problems. They are closely related to traffic map planning and can also be combined with VRP and TSP to form more complex optimization problems. They can theoretically be solved using the same RL framework by changing the corresponding Markov decision process.

5.4. Experimental data and evaluation indicators

The data sources in combinatorial optimization problems on graphs can be roughly summarized in the following aspects:

(1) Graph data in practical applications, such as relevant data in the natural environment, i.e., logistics and transportation, including the coordinated position of each delivery point, the amount of goods demanded, the number of nodes, the number of vehicles, etc.
(2) In experiments, the data sets used to train and test the effect of the model, often generated data such as TSP instances and VRP instances. For small-scale graph instances, we can employ a solution solver to obtain the optimal solution set.
(3) Trajectories generated during RL training, or data generated through self-play [130–133].
(4) Measurement indicators based on deep reinforcement methods can be summarized as follows:

Time. The training time is a vital indicator for using deep RL methods, such as the time required to train the policy network and value network, or the time required for the trained model to do the prediction task in an instance.

Hyperparameters. For example, the learning rate of gradient descent, the number of layers of the neural network, the number of neurons, and the batch size will significantly impact the effect of the model. The problem of over-fitting should also be considered.

Quality. The quality of a model or solution used to solve combinatorial optimization problems can be averaged over the entire set of test cases using the approximation ratio of each solution relative to the best solution [70]. The approximation ratio of a solution S to a problem instance G can be defined as: $R(S,G) = \max(\frac{OPT(G)}{c(h(S))}, \frac{c(h(S))}{OPT(G)})$, where $c(h(S))$ is the target value of the solution, and $OPT(G)$ is the value of the optimal solution of the problem instance G.

Generalization Ability. Can the model be generalized to larger problem instances, or can it be used to solve other combinatorial optimization problems of similar types? Because most combinatorial optimization problems on graphs have similar structures, one of the most important aspects of deep RL over classical exact or heuristic algorithms is generalization ability.

Effectiveness. General methods based on deep RL are composed of several neural network structures, so we can employ ablation research to verify the effectiveness of each component.

Decoding Strategy. Methods based on deep RL can also be combined with heuristic methods. For example, a greedy algorithm may be used in decoding, and the node with the highest

Table 1
Typical applications of deep learning and RL in combinatorial optimization problems on graphs.

NP-hard problems	Deep Learning Models + RL + Heuristic Search	Learning type
TSP	Pointer Network + Supervised Learning + Beam Search [104]	Supervised, Approximation
	Structure2vec + Q-Learning + Greedy [70]	Model-free, Value-based
	GPN+Hierarchical RL+Sampling/Greedy [135]	Model-free, Policy-based
	RNN with attention + REINFORCE + greedy/Beam Search [116]	Model-free, Policy-based
	Transformer + REINFORCE + Greedy [73]	Model-free, Policy-based
	Pointer network + Actor–Critic + Sampling/Active Search [115]	Model-free, Policy-based
	Sinkhorn layer+DDPG+meta-heuristic [136]	Model-free, Policy-based
	Feedforward Nets + REINFORCE + Transition Matrix [137]	Model-free, Actor–Critic
	GAT+deep Q-learning and PPO [138]+CP [139]	Model-free, Actor–Critic
	RNN with attention + REINFORCE + 2-opt [140]	Model-free, Actor–Critic
	GCN + Supervised Learning + Greedy/Beam Search [141]	Supervised, Approximation
VRP	RNN with attention + REINFORCE + Greedy/Beam Search [116]	Model-free, Policy-based
	LSTM + Q-Actor–Critic + rule-based rewriter [121]	Model-free, Policy-based
	Transformer + REINFORCE + Greedy Rollout Baseline [73]	Model-free, Policy-based
	Neural Network + REINFORCE + rule-based controller [122]	Model-free, Policy-based
	GAT and GRU + PPO + Beam Search [142]	Model-free, Actor–Critic
	GAT + REINFORCE + Greedy [143]	Model-free, Policy-based
	GCN and MLP+REINFORCE + Greedy [117]	Model-free, Policy-based
MVC	Structure2vec + Q-Learning + Greedy [70]	Model-free, Value-based
	DQN and Imitation learning [144]	Model-free, Value-based
	GNN + REINFORCE + Local search [145]	Model-free, Policy-based
	GCN + Supervised Learning + Guided Tree Search [126]	Model-based, Given model
	GCN + Supervised Learning and Q-Learning + Greedy [146]	Model-free, Value-based
MIS	GCN + Supervised Learning + Guided Tree Search [126]	Model-based, Given model
	GCN + AlphaGo Zero + MCTS [130]	Model-based, Given model
SAT	MPNN + Supervised Learning [147]	Supervised, Approximation
	GNN + REINFORCE + Local search [145]	Model-free, Policy-based
	GCN + Supervised Learning + Guided Tree Search [126]	Model-based, Given model
MC	GNN + REINFORCE + Local search [145]	Model-free, Policy-based
	GCN + Supervised Learning + Guided Tree Search [126]	Model-based, Given model
MaxCut	Structure2vec + Q-Learning + Greedy [70]	Model-free, Value-based
	MPNN + DQN + Greedy [148]	Model-free, Value-based
	CNN and GRU + PPO and Transfer learning [149]	Model-free, Actor–Critic

Note this contains three parts: deep learning, RL, and heuristic search algorithms. The learning types in the table indicate the type of RL to which the corresponding work belongs, where "supervised, approximation" means that the method only uses supervised learning, and "actor–critic" means that the method is based on the actor–critic framework. This table is referred to adapted from [136], from which we have selected and supplemented some appropriate methods.

probability will be selected as the next destination in each decoding step. Beam search (BS) tracks the most likely paths and then selects the one with the highest reward. There are other heuristics such as sampling, active search, etc., which require some baselines for comparison, which is usually based on the tour length of the path searched by the agent [70,115,117,135].

Efficiency. Parameter adjustments of network architectures significantly impact training accuracy and resource utilization, such as GPU or TPU utilization. The implementation framework (e.g., TensorFlow or Pytorch) is also worth considering.

Robustness. A model is robust if it runs multiple times, and its results fluctuate only within a small range.

Upper Limit and Lower Limit. The upper limit and lower limit are the best and worst solutions, respectively, for a combinatorial optimization problem.

There may be more detailed evaluation indicators, which need to be designed according to motivation and purpose. Above are some universal evaluation indices and experimental methods we can choose from to measure the efficiency and effectiveness of a model.

5.5. Performance of different methods

Here we summarize the recent experimental results of usual deep reinforcement learning-based methods and heuristic algorithms, which include the following aspects: (1) their performance on small-scale TSP instances (TSP20, TSP50, and TSP100), i.e., the length of the shortest path obtained and the time spent (Table 2); (2) the length of the shortest path they can obtain on large TSP instances (TSP250, TSP500, TSP750, and TSP1000) and the time they take to obtain it (Table 3); (3) the length of the shortest path they can obtain on the instances on CVRP (CVRP20, CVRP50, and CVRP100) and the time they take to obtain it (Table 4); (4) the length of the shortest path they can obtain on large-scale VRP instances (VRP100, VRP200, and VRP400) and the gap between the length of the shortest path and the absolute optimal solution (Table 5); (5) their approximate ratios to the optimal solution on a random TSP instance (Fig. 10); (6) a visual comparison of their learning curves (Fig. 11). The experimental results of different methods are drawn from recent papers on representative learning-based methods, such as AM [73], S2V-DQN [70], GCN-NPEC [117], GPN [135], PN-AC [115], PN [104], PRL [116], L2I [122], and GCN-NPEC (Node) [117].

From the experimental results, we can derive the following phenomena and facts:

(1) the performance of solution solvers (such as Concorde, LKH3, and Gurobi) based on heuristic algorithms still maintain a high level of accuracy, especially on small scale TSP and VRP instances. Learning-based approaches are constantly approaching the optimal solutions on a small scale, more so than traditional heuristic algorithms (such as Nearest Insertion, Random Insertion, Nearest Neighbor, etc.).

(2) Learning-based methods have more advantages on larger TSP and VRP instances, especially in less time than solution solvers and traditional heuristics.

Table 2
Performance of different algorithms on small scale TSP instances.

Method	TSP20		TSP50		TSP100	
	Obj.	Time	Obj.	Time	Obj.	Time
Concorde	3.84	(1 m)	5.70	(2 m)	7.76	(3 m)
LKH3	3.84	(18 s)	5.70	(5 m)	7.76	(21 m)
Gurobi	3.84	(7 s)	5.70	(2 m)	7.76	(17 m)
Nearest Insertion	3.84	(18 s)	6.78	(2 s)	9.46	(6 s)
Random Insertion	3.85	–	6.13	(1 s)	8.52	(3 s)
Farthest Insertion	3.93	(1 s)	6.01	(2 s)	8.35	(7 s)
Nearest Neighbor	4.50	(0 s)	7.00	(0 s)	9.68	(0 s)
PN [104]	3.88	–	7.66	–	–	–
PN-AC [115]	3.89	–	5.95	–	8.30	–
S2V-DQN [70]	3.89	–	5.99	–	8.31	–
AM [73]	3.85	(0 s)	5.80	(2 s)	8.12	(6 s)

Table 3
Performance of different algorithms on larger scale TSP instances.

Method	TSP250	TSP500	TSP750	TSP1000
	Obj. Time	Obj. Time	Obj. Time	Obj. Time
Concorde	11.89, 1894 s	16.55, 13902 s	20.10, 32993 s	23.11, 47804 s
LKH3	11.893, 9792 s	16.542, 23070 s	20.129,36840 s	23.130, 50680 s
OR Tools	12.289, 5000 s	17.449, 5000 s	22.395, 5000 s	26.477, 5000 s
Nearest Insertion	14.928, 25 s	20.791, 60 s	25.219, 115 s	28.973, 136 s
2-opt	13.253, 303 s	18.600, 1363 s	22.668, 3296 s	26.111, 6153 s
Farthest Insertion	13.026, 33 s	18.288, 160 s	22.342, 454 s	25.741, 945 s
PN [104]	14.249, 29 s	21.409, 280 s	27.382, 782 s	32.714, 3133 s
GPN [135]	12.942, 214 s	18.358, 974 s	22.541, 2278 s	26.129, 4410 s
S2V-DQN [70]	13.079, 476 s	18.428, 1508 s	22.550, 3182 s	26.046, 5600 s
AM [73]	14.032, 2 s	24.789, 14 s	28.281, 42 s	34.055, 136 s

Table 4
Performance of different algorithms on small scale CVRP instances.

Method	CVRP20	CVRP50	CVRP100
	Obj. Time	Obj. Time	Obj. Time
Gurobi	6.10, –	–, –	–, –
LKH3	6.14, 2 h	10.38, 7 h	15.65, 13 h
Random CW	6.81, –	12.25, –	18.96, –
Random Sweep	7.08, –	12.96, –	20.33, –
OR Tools	6.43, –	11.31, –	17.16, –
AM (greedy) [73]	6.40, 1 s	10.98, 3 s	16.80, 8 s
AM (sampling) [73]	6.25, 6 m	10.62, 28 m	16.23, 2 h
PRL [116]	6.40, –	11.15, –	16.96, –
L2I [122]	6.12, 12 m	10.35, 17 m	15.57, 24 m

Table 5
Performance of different algorithms on larger VRP instances.

Method	VRP100		VRP200		VRP400	
	Obj	Gap	Obj	Gap	Obj	Gap
OR-Tools	11.348	0.00%	17.995	0.00%	26.095	0.00%
PRL [116]	11.907	4.93%	19.521	8.48%	28.846	10.54%
AM [73]	11.746	3.51%	18.697	3.90%	27.143	4.02%
GCN-NPEC [117]	11.186	−1.43%	17.516	−2.66%	25.876	−0.84%

(3) From the perspective of solution quality, learning-based methods are also higher than traditional heuristic methods because the results obtained by the former method are closer to the optimal solutions.

(4) We can make the models' learning converge more quickly by introducing more diversified training methods (combining supervised learning and reinforcement learning).

(5) There is still a lot of room for improvement in learning-based methods, especially in the efficient processing of large-scale graph data. Deep reinforcement learning makes models have more generalization ability and the ability to simultaneously solve multiple types of problems, which is impossible for solution solvers.

5.6. Challenges encountered by deep RL in solving combinatorial optimization problems

Applying deep RL methods (deep learning, RL, and fusions with heuristic methods) to solve combinatorial optimization problems on graphs is a research hotspot and trend in recent years. Compared with traditional methods, such as exact algorithms, approximation algorithms, etc., learning-based methods have better robustness and generalization ability and are entirely data-driven and self-driven. By sorting out previous related work and analyzing experiments, we also summarize the difficulties of machine learning for solving combinatorial optimization problems and the challenges brought about by the characteristics of machine learning itself.

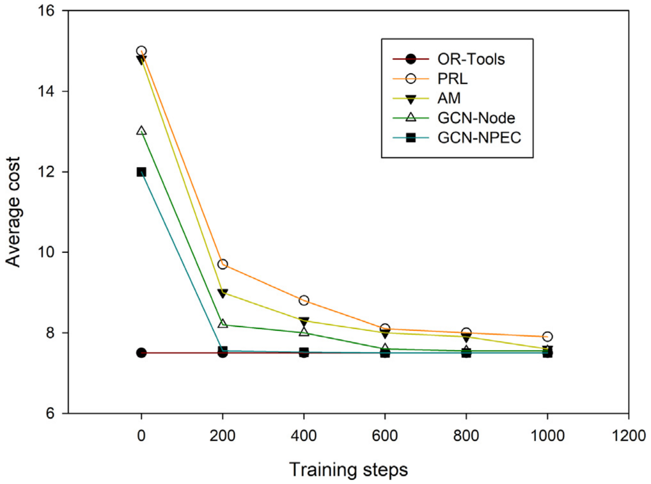

Fig. 11. Comparison of learning curves of different methods.

5.6.1. Feasibility

We have illustrated how to apply machine learning to output solutions to combinatorial optimization problems directly. Rather than learning solutions, machine learning algorithms are essentially learning heuristics. According to the definition of heuristics, "An algorithm constructed intuitively or empirically gives a feasible solution for each instance of a combinatorial optimization problem to be solved at an acceptable cost (computational time and space). The degree of deviation from the optimal solution cannot generally be predicted". The learned algorithm is not guaranteed to be optimal, nor is it guaranteed to be feasible. We do not know how far the heuristic's output is from the optimal solution or whether it strictly follows the problem's constraints. In fact, in routing problems, most of the learning-based methods are focusing on elementary ones with fewer constraints, as most of these constraints are represented by masks. However, real-world routing problems have many complex constraints, and applying deep RL to solve them is challenging. Finding feasible solutions is not easy because the problems are NP-hard. It is also a challenge in machine learning, especially when using neural networks, because the neural structure for gradient descent training must be carefully designed not to destroy differentiability. Network architectures are usually complex (such as pointer networks, GNNs, transformers, etc.), equivalent to solving a complex problem with an inherently complex method. Still, the design of a classic heuristic method is relatively more straightforward. Therefore, the feasibility of machine learning methods in solving optimization problems in actual conditions and practical applications needs to be further evaluated.

5.6.2. Scalability & complexity

Whether deep RL models can be extended to a larger scale is also a challenge. For example, a model can be trained on instances of 50 nodes, but there are generalization issues if trained on larger instances (such as instances of 1000 or 10 000 nodes). Almost all papers dealing with TSP through machine learning and trying to solve larger instances believe that with the increase of scale, performance will decline [70,73,116,141,143]. To solve this problem, we can just try to learn larger instances, but it may become a problem of computation and generalization. In addition to elementary machine learning models and strong assumptions about data distribution, it is impossible to know the computational and sample complexity, i.e., the number of observations required to learn, because we do not know the actual data generation distribution.

Table 6
Different methods run on TSP100 Instances.

Method	Complexity	Runtime	Speedup	Approximation
Gurobi	–	3220	2752.1	1
Concorde	–	254.1	217.2	1
Christofide	$O(n^3)$	5002	4275.2	1.029
LKH	$O(n^{2.2})$	2879	2460.7	1
2-opt	$O(n^2)$	30.08	25.7	1.097
Farthest	$O(n^2)$	8.35	7.1	1.075
Nearest	$O(n^2)$	9.35	8	1.245
S2V-DQN [70]	$O(n^2)$	61.72	52	1.084
GPN [135]	$O(nlog(n))$	1.537	1.3	1.086
GATM [143]	$O(n)$	1.17	1	1.074

Note that their time complexity, running time (ms), speedup factor relative to GATM [136], and approximation ratio relative to the optimal solution.

Note that their time complexity, running time (ms), speedup factor relative to GATM [136], and approximation ratio relative to the optimal solution.

Unlike traditional algorithms that only consider the time complexity (Table 6), learning-based methods must consider both the space complexity and the time complexity. The time complexity determines the training or prediction time of the model. If the time complexity is too high, the model training and prediction will consume too much time, and it is impossible to verify the algorithm and improve the model quickly. It is impossible to make a rapid prediction. The space complexity determines the number of parameters of the model. The more parameters in a model, the greater the amount of data required to train the model. However, data sets in real life are usually not too large, making it easier for a model's training to overfit.

5.6.3. Data

Access to data is a challenge because most previous work used random functions to generate data to train the policy network. We can also collect data to train the required policy and expect that the policy can be generalized on corresponding instances of the given program. Nonetheless, how can we actively train policies for problems we do not yet know about when we do not have specific applications with targeted historical data? We first need to define what instances we want to generalize to. In addition, most relevant works on TSP and VRP use a uniform distribution to train the model. What if the node locations do not follow any distribution or their distributions are unknown? In this case, training a robust deep RL model is challenging, as it is not a matter of simply using a dataset of mixed distributions. For models using supervised learning, the acquisition of large amounts of expensive labeled data is also a challenge.

How to represent data is not easy, but it has a significant impact on learning. Representation learning methods need to be expressive enough and concise enough to be used repeatedly without too much computation. When the number of layers of a traditional GNN reaches a particular scale, there will be problems with suspended animation and over-smoothing, so in general, GNNs are only set to two layers [150]. More importantly, when the size of the graph becomes larger, a traditional GNN model will be too heavy due to a large number of parameters, which will degrade the performance and even fail to be trained.

Table 7 demonstrates the performance of three learning-based methods on large-scale synthetic data sets, in which S2V-DQN [70] has no results on larger data sets and may even collapse [146]. GCN-TreeSearch applies a graph reduction technique to reduce a graph to a smaller one to handle larger graphs [126], and applies supervised learning to train the model iteratively,

Table 7
The performance comparison of S2V-DQN.

Graph	S2V-DQN [70]	GCOMB [146]	GCN-TreeSearch [126]
BA-10k	1935	4625	4633
BA-20k	2464	6533	6533
BA-50k	4138	10398	10378
BA-100k	–	14473	14484
BA-500k	–	30427	30626
Gowalla	–	76793	76344
Brightkite	–	14759	14634

S2V-DQN is a reinforcement learning method based on a GNN, GCN-TreeSearch, a method based on supervised learning and a traditional search algorithm, and GCOMB, a method combining reinforcement learning and supervised learning, in MVC on synthetic data sets (Barabasi-Albert [146]).

leading to inefficiency [146]. GCOMB combines the strengths of both with a two-stage framework of supervised learning followed by reinforcement learning to process billions of levels of graph data [146]. In the stage of supervised learning, it applies a GCN to learn more "appropriate" nodes as candidate solutions. In the stage of reinforcement learning, the solution of the problem is selected from the candidate solutions. In practice, GCOMB [146] eliminates most of the "useless" nodes of the graph in the first stage, reducing the number of alternative nodes considered in the computation-intensive stage of reinforcement learning to achieve acceleration. However, methods that extend models to large-scale data are essentially preprocessing input graphs using traditional methods rather than innovatively learning methods to process large-scale graphs.

S2V-DQN is a reinforcement learning method based on a GNN, GCN-TreeSearch, a method based on supervised learning and a traditional search algorithm, and GCOMB, a method combining reinforcement learning and supervised learning, in MVC on synthetic data sets (Barabasi–Albert [146]).

5.6.4. Modeling & training

Generally, in deep learning, we will have some prior knowledge of some given problems. For example, a convolutional neural network (CNN) is a structure that is easier to train on image data, and a recurrent neural network (RNN) is most suitable for sequence data. The problems studied in combinatorial optimization are different from the natural signals that deep learning is best at. In other words, the neural network architecture suitable for combinatorial optimization problems may be very different from current mainstream deep learning models.

Additionally, deep neural networks often require billions of parameters, more than a mainframe computer can handle, or require expensive hardware support and long computing times. Current machine learning algorithms are equivalent to energy optimization in terms of training difficulty, and the stochastic gradient descent method is generally applied for optimization. However, due to an excessive number of layers in deep neural networks, gradient disappearance and gradient explosion often occur, so the it is difficult for the training process to converge. In addition, the lack of a theoretical basis for deep learning also leads to a failure to guarantee convergence.

5.7. Future research directions

Through comprehensive analysis of the existing methods, we also identify possible future research directions, which in some respects correspond to the challenges of applying deep reinforcement learning to combinatorial optimization.

5.7.1. More appropriate and effective training methods

As deep RL evolves, there will be novel training algorithms or paradigms to replace the existing ones, such as automated deep learning [151], meta-learning [152,153], generative adversarial imitation learning [154,155], inverse reinforcement learning [156,157], self-supervised learning [158,159], MuZero [82], etc. They are likely to improve training efficiency and have less dependence on data. There will be state-of-the-art models newly proposed, such as BERT [160], GPT [161], and other modern translation models or GNNs that will achieve state-of-the-art results. If we properly improve them and apply them to combinatorial optimization, will they also achieve promising results?

5.7.2. Combinations with traditional algorithms

As mentioned above, existing learning-based methods are essentially learning heuristics. If we combine deep RL with traditional heuristics (especially solution solvers) or search algorithms, will it become easier for agents to explore the tremendous combinatorial space?

In addition to the pruning strategy described above, we can also employ more classical data mining algorithms to preprocess the graph data to a manageable size from a large graph. We can also incorporate graphs into the training process, including greedy probability distributions, budget constraints [146], etc. Learning heuristics can be used in many fields or tasks, such as community discovery [162], knowledge graph reasoning, influence maximization [146,163], etc., which positively promotes the growth and cross-integration of multiple fields.

5.7.3. Universality

We should continue to focus on strengthening the fundamental advantage of deep reinforcement learning, which is universality. A single approach to many different problems is appealing because it will bring huge benefits and eliminate inefficient manual work, which is perhaps the core pursuit of artificial intelligence. It is essential to apply learning-based methods to large-scale practical scenarios; this is also the driving force for combinatorial optimization research. The application of artificial intelligence in combinatorial optimization also plays a positive role in developing artificial intelligence itself. When there is a general algorithm that can solve all combinatorial optimization problems, the era of general artificial intelligence may come.

5.7.4. Explainability

There are many theoretical studies on combinatorial optimization (CO). Nevertheless, there are relatively few theoretical studies on applying deep reinforcement learning to CO. As stated earlier, we do not know precisely how far resulting solutions are from the optimal solution and whether they strictly satisfy the constraints. Therefore, it is necessary to conduct theoretical research on applying deep reinforcement learning to combinatorial optimization problems and study models that can satisfy more complex constraints. It is also a promising direction to exploit the methods of mathematics, operations research, computer science, and other disciplines to conduct basic theoretical research to enhance the explainability of these models.

6. Concluding remarks

We summarized deep reinforcement learning applied to graph combinatorial optimization in recent years, and described some learning models and algorithms in detail. We (1) clarified the significance and purpose of such a related research, (2) emphasized the fundamental motivation of applying deep reinforcement learning, (3) analyzed and summarized the performance comparison between different methods, and (4) sorted out the challenges faced by learning-based methods and future research directions. The fundamental advantage of learning-based methods over traditional algorithms is the ability to automatically learn heuristics, extend to large-scale instances, and generalize to other similar problems. The combination of state-of-the-art learning models with classical algorithms will be mainstream in the near future.

Declaration of competing interest

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

Acknowledgments

The authors thank Kenneth Lai, MA, Yun Xiong, PhD, and Yangyong Zhu, PhD, for valuable comments on the early versions. The content is solely the responsibility of the authors.

References

[1] V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, R. Addanki, T. Hapuarachchi, T. Keck, J. Keeling, P. Kohli, I. Ktena, Y. Li, O. Vinyals, Y. Zwols, Solving mixed integer programs using neural networks, pp. 1–57, 2020, http://arxiv.org/abs/2012.13349.

[2] E. Balas, The prize collecting traveling salesman problem: II. Polyhedral results, Networks 25 (1995) 199–216, http://dx.doi.org/10.1002/net.3230250406.

[3] S. Goyal, A survey on travelling salesman problem, Midwest Instr. Comput. Symp. (2010) 1–9.

[4] C.H. Häll, H. Andersson, J.T. Lundgren, P. Värbrand, I. Nedregård, Paolo. Toth, Daniele. Vigo, I. Nedregård, O. Gurobi, E. Pimpler, J. Laflaquiere, U.M. Sundar, C. Yang, J.F. Cordeau, C.H. Häll, H. Andersson, J.T. Lundgren, P. Värbrand, M. Posada, H. Andersson, C.H. Häll, Vehicle routing, Public Transp. (2006) 573–586, http://dx.doi.org/10.1007/s12469-008-0006-1.

[5] B. Eksioglu, A.V. Vural, A. Reisman, The vehicle routing problem: A taxonomic review, Comput. Ind. Eng. 57 (2009) 1472–1483, http://dx.doi.org/10.1016/j.cie.2009.05.009.

[6] M. Gansterer, R.F. Hartl, Collaborative vehicle routing: A survey, European J. Oper. Res. 268 (2018) 1–12, http://dx.doi.org/10.1016/j.ejor.2017.10.023.

[7] C. Liu, G. Kou, X. Zhou, Y. Peng, H. Sheng, F.E. Alsaadi, Time-dependent vehicle routing problem with time windows of city logistics with a congestion avoidance approach, Knowledge-Based Syst. 188 (2020) 104813, http://dx.doi.org/10.1016/j.knosys.2019.06.021.

[8] Q. Gu, Q. Wang, X. Li, X. Li, A surrogate-assisted multi-objective particle swarm optimization of expensive constrained combinatorial optimization problems, Knowledge-Based Syst. 223 (2021) 107049, http://dx.doi.org/10.1016/j.knosys.2021.107049.

[9] A. Hamzadayı, A. Baykasoğlu, Ş. Akpınar, Solving combinatorial optimization problems with single seekers society algorithm, Knowl.-Based Syst. 201–202 (2020) 106036, http://dx.doi.org/10.1016/j.knosys.2020.106036.

[10] Y. Lecun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444, http://dx.doi.org/10.1038/nature14539.

[11] S.S. Mousavi, M. Schukat, E. Howley, Deep reinforcement learning: An overview, Lect. Notes Netw. Syst. 16 (2018) 426–440, http://dx.doi.org/10.1007/978-3-319-56991-8_32.

[12] A. Mor, M.G. Speranza, Vehicle routing problems over time: a survey, 4or 18 (2020) 129–149, http://dx.doi.org/10.1007/s10288-020-00433-2.

[13] M. Han, Y. Wang, A survey for vehicle routing problems and its derivatives, IOP Conf. Ser. Mater. Sci. Eng. 452 (2018) http://dx.doi.org/10.1088/1757-899X/452/4/042024.

[14] D. Rojas Viloria, E.L. Solano-Charris, A. Muñoz Villamizar, J.R. Montoya-Torres, Unmanned aerial vehicles/drones in vehicle routing problems: a literature review, Int. Trans. Oper. Res. 28 (2021) 1626–1657, http://dx.doi.org/10.1111/itor.12783.

[15] M.I. Jordan, T.M. Mitchell, Machine learning: Trends, Perspectives, and Prospects, Nature 349 (2015).

[16] D.M. Blei, P. Smyth, Science and data science, Proc. Natl. Acad. Sci. U. S. A 114 (2017) 8689–8692, http://dx.doi.org/10.1073/pnas.1702076114.

[17] A. Sherstinsky, Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network, Phys. D Nonlinear Phenom. 404 (2020) 1–43, http://dx.doi.org/10.1016/j.physd.2019.132306.

[18] S. Chehreh Chelgani, B. Shahbazi, E. Hadavandi, Support vector regression modeling of coal flotation based on variable importance measurements by mutual information method, Meas. J. Int. Meas. Confed. 114 (2018) 102–108, http://dx.doi.org/10.1016/j.measurement.2017.09.025.

[19] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: 3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc. 2015, pp. 1–14.

[20] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, Adv. Neural Inf. Process. Syst. 4 (2014) 3104–3112.

[21] Y. Li, R. Zemel, M. Brockschmidt, D. Tarlow, Gated graph sequence neural networks, in: 4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc., 2016, pp. 1–20.

[22] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. 2016 (2016) 770–778, http://dx.doi.org/10.1109/CVPR.2016.90.

[23] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: 32nd Int. Conf. Mach. Learn. ICML 2015, vol. 1, 2015, pp. 448–456.

[24] Y. Wu, K. He, Group normalization, Int. J. Comput. Vis. 128 (2020) 742–755, http://dx.doi.org/10.1007/s11263-019-01198.

[25] T. Bachlechner, B.P. Majumder, H.H. Mao, G.W. Cottrell, J. McAuley, ReZero Is all you need: Fast convergence at large depth, pp. 1–11, 2020, http://arxiv.org/abs/2003.04887.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting nitish, J. Mach. Learn. Res. 15 (2018) 7642–7651, http://dx.doi.org/10.1109/CVPR.2018.00797.

[27] L. Prechelt, Early stopping - But when? in: Lect. Notes Comput. Sci., in: (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 7700 LECTU, 2012, pp. 53–67, http://dx.doi.org/10.1007/978-3-642-35289-8_5.

[28] D. Zhang, J. Yin, X. Zhu, C. Zhang, Network representation learning: A survey, IEEE Trans. Big Data 6 (2018) 3–28, http://dx.doi.org/10.1109/tbdata.2018.2850013.

[29] R.L. Murphy, B. Srinivasan, V. Rao, B. Ribeiro, Relational pooling for graph representations, in: 36th Int. Conf. Mach. Learn. ICML 2019-June, 2019, pp. 8192–8202.

[30] B. Scholkopf, F. Locatello, S. Bauer, N.R. Ke, N. Kalchbrenner, A. Goyal, Y. Bengio, Toward causal representation learning, Proc. IEEE. 109 (2021) 612–634, http://dx.doi.org/10.1109/JPROC.2021.3058954.

[31] H. Cai, V.W. Zheng, K.C.C. Chang, A comprehensive survey of graph embedding: Problems, techniques, and applications, IEEE Trans. Knowl. Data Eng. 30 (2018) 1616–1637, http://dx.doi.org/10.1109/TKDE.2018.2807452.

[32] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S.G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A.P. Badia, K.M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, D. Hassabis, Hybrid computing using a neural network with dynamic external memory, Nature 538 (2016) 471–476, http://dx.doi.org/10.1038/nature20101.

[33] W.L. Hamilton, Le R 600, Butane, Rev. Prat. Du Froid Du Cond. d'Air 59 (2003).

[34] Y. Huang, Deep Q-networks, Deep Reinf. Learn. Fundam. Res. Appl. 1 (2020) 135–160, http://dx.doi.org/10.1007/978-981-15-4095-0_4.

[35] O. Vinyals, I. Babuschkin, W.M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D.H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J.P. Agapiou, M. Jaderberg, A.S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T.L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, D. Silver, Grandmaster level in StarCraft II using multi-agent reinforcement learning, Nature 575 (2019) 350–354, http://dx.doi.org/10.1038/s41586-019-1724-z.

[36] L. Xin, W. Song, Z. Cao, J. Zhang, Step-wise deep learning models for solving routing problems, IEEE Trans. Ind. Informatics 17 (2021) 4861–4871, http://dx.doi.org/10.1109/TII.2020.3031409.

[37] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks, ArXiv. pp. 1–40, 2018, http://arxiv.org/abs/1806.01261.

[38] M. Li, Y. Yang, C. Wang, Z. Qin, Z. Gong, G. Wu, Y. Jiao, J. Wang, J. Ye, Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning, in: Web Conf. 2019 - Proc. World Wide Web Conf. WWW 2019, 2019, pp. 983–994, http://dx.doi.org/10.1145/3308558.3313433.

[39] C. Gutierrez, J.F. Sequeda, Knowledge graphs: A tutorial on the history of knowledge graph's main ideas, Int. Conf. Inf. Knowl. Manag. Proc. (2020) 3509–3510, http://dx.doi.org/10.1145/3340531.3412176.

[40] B. Samanta, A. De, G. Jana, P.K. Chattaraj, N. Ganguly, M.G. Rodriguez, NEVAE: A Deep generative model for molecular graphs, in: 33rd AAAI Conf. Artif. Intell. AAAI 2019, 31st Innov. Appl. Artif. Intell. Conf. IAAI 2019 9th AAAI Symp. Educ. Adv. Artif. Intell. EAAI 2019, 2019, pp. 1110–1117, http://dx.doi.org/10.1609/aaai.v33i01.33011110.

[41] J. You, B. Liu, R. Ying, V. Pande, J. Leskovec, Graph convolutional policy network for goal-directed molecular graph generation, Adv. Neural Inf. Process. Syst. 2018 (2018) 6410–6421.

[42] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P.S. Yu, A comprehensive survey on graph neural networks, IEEE Trans. Neural Networks Learn. Syst. 32 (2021) 4–24, http://dx.doi.org/10.1109/TNNLS.2020.2978386.

[43] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc., 2019, pp. 1–14.

[44] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, Y. Bengio, Graph attention networks, in: 6th Int. Conf. Learn. Represent. ICLR 2018 - Conf. Track Proc. 2018, pp. 1–12.

[45] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, AI Open 1 (2020) 57–81, http://dx.doi.org/10.1016/j.aiopen.2021.01.001.

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. 07-12-June, 2015, pp. 1–9, http://dx.doi.org/10.1109/CVPR.2015.7298594.

[47] R. Autoencoders, W. Yu, C. Zheng, W. Cheng, C.C. Aggarwal, D. Song, B. Zong, H. Chen, W. Wang, Learning deep network representations with adversarially, Kdd (2018) 2663–2671.

[48] Z. Zhang, P. Cui, W. Zhu, Deep learning on graphs: A survey, IEEE Trans. Knowl. Data Eng. (2020) http://dx.doi.org/10.1109/tkde.2020.2981333, 1–1.

[49] S. Madjiheurem, L. Toni, Representation learning on graphs: A reinforcement learning application, in: AISTATS 2019-22nd Int. Conf. Artif. Intell. Stat., 89, 2020.

[50] A. Grover, J. Leskovec, Node2vec: Scalable feature learning for networks, in: Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. 13-17-Augu, 2016, pp. 855–864, http://dx.doi.org/10.1145/2939672.2939754.

[51] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min, 2014, pp. 701–710, http://dx.doi.org/10.1145/2623330.2623732.

[52] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, 1st Int. Conf. Learn. Represent. ICLR 2013 - Work. Track Proc., 2013, pp. 1–12.

[53] J. Tang, M. Qu, LINE: Large-scale Information Network Embedding, in: Int. World Wide Web Conf. Com_mittee, 2015, pp. 1067–1077.

[54] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. 13-17-Augu, 2016, pp. 1225–1234, http://dx.doi.org/10.1145/2939672.2939753.

[55] K. Xu, S. Jegelka, W. Hu, J. Leskovec, How powerful are graph neural networks?, in: 7th Int. Conf. Learn. Represent. ICLR 2019, 2019, 1–17.

[56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Adv. Neural Inf. Process. Syst., 2017, pp. 5999–6009.

[57] D. Bahdanau, K.H. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: 3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc., 2015, pp. 1–15.

[58] R. Sennrich, B. Haddow, A. Birch, Neural machine translation of rare words with subword units, in: 54th Annu. Meet. Assoc. Comput. Linguist. ACL 2016 - Long Pap, 3, 2016, pp. 1715–1725, http://dx.doi.org/10.18653/v1/p16-1162.

[59] J. Cheng, L. Dong, M. Lapata, Long short-term memory-networks for machine reading, in: EMNLP 2016 - Conf. Empir. Methods Nat. Lang. Process. Proc, 2016, pp. 551–561, http://dx.doi.org/10.18653/v1/d16-1053.

[60] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, in: 34th Int. Conf. Mach. Learn. ICML 2017, vol. 3, 2017, pp. 2053–2070.

[61] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and deep locally connected networks on graphs, in: 2nd Int. Conf. Learn. Represent. ICLR 2014 - Conf. Track Proc., 2014, pp. 1–14.

[62] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: Adv. Neural Inf. Process. Syst., 2015, pp. 2224–2232.

[63] K.T. Schütt, F. Arbabzadah, S. Chmiela, K.R. Müller, A. Tkatchenko, Quantum-chemical insights from deep tensor neural networks, Nature Commun. 8 (2017) 6–13, http://dx.doi.org/10.1038/ncomms13890.

[64] B. Bai, Y. Liu, C. Ma, G. Wang, G. Yan, K. Yan, M. Zhang, Z. Zhou, Graph neural network, Sci. Sin. Math. 50 (2020) 367–384, http://dx.doi.org/10.1360/N012019-00133.

[65] D. Zhu, D. Wang, P. Cui, W. Zhu, Deep variational network embedding in wasserstein space, Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (2018) 2827–2836, http://dx.doi.org/10.1145/3219819.3220052.

[66] N. Vesselinova, R. Steinert, D.F. Perez-Ramirez, M. Boman, Learning combinatorial optimization on graphs: A survey with applications to networking, IEEE Access. 8 (2020) 120388–120416, http://dx.doi.org/10.1109/ACCESS.2020.3004964.

[67] K.A. Smith, Neural networks for combinatorial optimization: A review of more than a decade of research, INFORMS J. Comput. 11 (1999) 15–34, http://dx.doi.org/10.1287/ijoc.11.1.15.

[68] J. Ye, J. Zhao, K. Ye, C. Xu, How to build a graph-based deep learning architecture in traffic domain: A survey, IEEE Trans. Intell. Transp. Syst. (2020) 1–21, http://dx.doi.org/10.1109/TITS.2020.3043250.

[69] J. Li, L. Xin, Z. Cao, A. Lim, W. Song, J. Zhang, Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning, IEEE Trans. Intell. Transp. Syst. 1 (2021) 1–10, http://dx.doi.org/10.1109/TITS.2021.3056120.

[70] H. Dai, E.B. Khalil, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, Adv. Neural Inf. Process. Syst. 2017 (2017) 6349–6359.

[71] H. Dai, B. Dai, L. Song, Discriminative embeddings of latent variable models for structured data, 33rd Int. Conf. Mach. Learn. ICML 2016 6 (2016) 3970–3986.

[72] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533, http://dx.doi.org/10.1038/nature14236.

[73] W. Kool, H. Van Hoof, M. Welling, Attention, learn to solve routing problems!, in: 7th Int. Conf. Learn. Represent. ICLR, 2019, pp. 1–25.

[74] A.P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, C. Blundell, Agent57: Outperforming the atari human benchmark, in: 37th Int. Conf. Mach. Learn. ICML 2020. PartF16814, 2020, pp. 484–494.

[75] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, Science (80-) 362 (2018) 1140–1144, http://dx.doi.org/10.1126/science.aar6404.

[76] K. Menda, Y.C. Chen, J. Grana, J.W. Bono, B.D. Tracey, M.J. Kochenderfer, D. Wolpert, Deep reinforcement learning for event-driven multi-agent decision processes, IEEE Trans. Intell. Transp. Syst. 20 (2019) 1259–1268, http://dx.doi.org/10.1109/TITS.2018.2848264.

[77] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: A methodological tour d'horizon, European J. Oper. Res. 290 (2021) 405–421, http://dx.doi.org/10.1016/j.ejor.2020.07.063.

[78] P.R.de O. da Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning, ArXiv, pp. 465–480, 2020, http://arxiv.org/abs/2004.01608.

[79] M.L. Littman, Reinforcement learning improves behaviour from evaluative feedback, Nature 521 (2015) 445–451, http://dx.doi.org/10.1038/nature14540.

[80] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of go with deep neural networks and tree search, Nature 529 (2016) 484–489, http://dx.doi.org/10.1038/nature16961.

[81] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge, Nature 550 (2017) 354–359, http://dx.doi.org/10.1038/nature24270.

[82] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, D. Silver, Mastering atari, go, chess and shogi by planning with a learned model, Nature 588 (2020) 604–609, http://dx.doi.org/10.1038/s41586-020-03051-4.

[83] C. Jin, Z. Allen-Zhu, S. Bubeck, M.I. Jordan, Is Q-learning provably efficient? Adv. Neural Inf. Process. Syst. 2018 (2018) 4863–4873.

[84] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533, http://dx.doi.org/10.1038/nature14236.

[85] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-Learning, in: 30th AAAI Conf. Artif. Intell. AAAI 2016, 2016, pp. 2094–2100.

[86] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, N. De Frcitas, Dueling Network Architectures for Deep Reinforcement Learning, in: 33rd Int. Conf. Mach. Learn. ICML 2016, 2016, pp. 2939–2947.

[87] W. Dabney, M. Rowland, M.G. Bellemare, R. Munos, Distributional reinforcement learning with quantile regression, in: 32nd AAAI Conf. Artif. Intell. AAAI 2018, 2018, pp. 2892–2901.

[88] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D.S. Deepmind, Rainbow: combining improvements in dqn, in: Thirty-Second AAAI Conf. Artif. Intell., 2018, pp. 3215–3222, https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/viewFile/17204/16680.

[89] M.E. Taylor, S. Whiteson, P. Stone, Temporal difference and policy search methods for reinforcement learning: An empirical comparison, Proc. Natl. Conf. Artif. Intell. 2 (2007) 1675–1678.

[90] M.P. Deisenroth, C.E. Rasmussen, PILCO: A model-based and data-efficient approach to policy search, in: Proc. 28th Int. Conf. Mach. Learn. ICML 2011, 2011, pp. 465–472.

[91] R.J. Willia, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (1992) 229–256, http://dx.doi.org/10.1023/A:1022672621406.

[92] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, A. Courville, R.L.J. Pineau, Y. Bengio, An actor-critic algorithm for sequence prediction, in: 5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc., 2017, pp. 1–17.

[93] S. Ivanov, A. D'yakonov, Modern deep reinforcement learning algorithms, ArXiv, 2019, http://arxiv.org/abs/1906.10025.

[94] T. Weng, J. Uesato, K. Xiao, S. Gowal, R. Stanforth, P. Kohli, Toward evaluating robustness of deep rein- forcement learning with continuous control, Iclr 1 (2020) 1–13.

[95] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in: 31st Int. Conf. Mach. Learn. ICML 2014, vol. 1, 2014, pp. 605–619.

[96] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Comput. Oper. Res. 34 (2007) 2403–2435, http://dx.doi.org/10.1016/j.cor.2005.09.012.

[97] K. Helsgaun, Effective implementation of the lin-kernighan traveling salesman heuristic, Eur. J. Oper. Res. 126 (2000) 106–130, http://dx.doi.org/10.1016/S0377-2217(99)00284-2.

[98] C.W. Chiang, W.P. Lee, J.S. Heh, A 2-opt based differential evolution for global optimization, Appl. Soft Comput. J. 10 (2010) 1200–1207, http://dx.doi.org/10.1016/j.asoc.2010.05.012.

[99] C. Blum, J. Puchinger, G.R. Raidl, A. Roli, Hybrid metaheuristics in combinatorial optimization: A survey, Appl. Soft Comput. J. 11 (2011) 4135–4151, http://dx.doi.org/10.1016/j.asoc.2011.02.032.

[100] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: A survey of the state of the art, J. Oper. Res. Soc. 64 (2013) 1695–1724, http://dx.doi.org/10.1057/jors.2013.71.

[101] J.J. Hopfield, D.W. Tank, Neural computation of decisions in optimization problems, Biol. Cybernet. 52 (1985) 141–152, http://dx.doi.org/10.1007/BF00339943.

[102] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, T. Adler, L. Gruber, M. Holzleitner, M. Pavlović, G.K. Sandve, V. Greiff, D. Kreil, M. Kopp, G. Klambauer, J. Brandstetter, S. Hochreiter, Hopfield networks is all you need, 2020, ArXiv. http://arxiv.org/abs/2008.02217.

[103] V. Mnih, N. Heess, A. Graves, K. Kavukcuoglu, Recurrent models of visual attention, Adv. Neural Inf. Process. Syst. 3 (2014) 2204–2212.

[104] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in: Adv. Neural Inf. Process. Syst. 2015-Janua, 2015, pp. 2692–2700, http://arxiv.org/abs/1506.03134.

[105] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, J. Dean, Outrageously large neural networks: the sparsely-gated mixture-of-experts layer, in: Int. Conf. Learn. Represent, 2017, 1–19.

[106] Y. Luo, J. Ji, X. Sun, L. Cao, Y. Wu, F. Huang, C.-W. Lin, R. Ji, Dual-level collaborative transformer for image captioning, 2021, http://arxiv.org/abs/2101.06462.

[107] W.U. Ahmad, N. Peng, K.W. Chang, Gate: graph attention transformer encoder for cross-lingual relation and event extraction, ArXiv, 2020.

[108] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, W. Zhang, Informer: beyond efficient transformer for long sequence time-series forecasting, ArXiv, 2020.

[109] Y. Hu, Y. Yao, W.S. Lee, A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs, Knowledge-Based Syst. 204 (2020) 106244, http://dx.doi.org/10.1016/j.knosys.2020.106244.

[110] J. Kotary, F. Fioretto, P. Van Hentenryck, B. Wilder, End-to-End constrained optimization learning: A survey, http://dx.doi.org/10.24963/ijcai.2021/610.

[111] R. Bai, X. Chen, Z.-L. Chen, T. Cui, S. Gong, W. He, X. Jiang, H. Jin, J. Jin, G. Kendall, J. Li, Z. Lu, J. Ren, P. Weng, N. Xue, H. Zhang, Analytics and machine learning in vehicle routing research, 2021, http://arxiv.org/abs/2102.10012.

[112] N. Mazyavkina, S. Sviridov, S. Ivanov, E. Burnaev, Reinforcement learning for combinatorial optimization: A survey, Comput. Oper. Res. 134 (2021) 1–12, http://dx.doi.org/10.1016/j.cor.2021.105400.

[113] L.C. Lamb, A. d'Avila Garcez, M. Gori, M.O.R. Prates, P.H.C. Avelar, M.Y. Vardi, Graph neural networks meet neural-symbolic computing: A survey and perspective, in: IJCAI Int. Jt. Conf. Artif. Intell. 2021-Janua, 2020, pp. 4877–4884, http://dx.doi.org/10.24963/ijcai.2020/679.

[114] Q. Cappart, D. Chételat, E.B. Khalil, A. Lodi, C. Morris, P. Veličković, Combinatorial optimization and reasoning with graph neural networks, 2021, pp. 4348–4355. http://dx.doi.org/10.24963/ijcai.2021/595.

[115] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, in: 5th Int. Conf. Learn. Represent. ICLR 2017 - Work. Track Proc, 2017, pp. 1–15.

[116] M. Nazari, A. Oroojlooy, M. Takáč, L.V. Snyder, Reinforcement learning for solving the vehicle routing problem, Adv. Neural Inf. Process. Syst. 2018 (2018) 9839–9849.

[117] L. Duan, Y. Zhan, H. Hu, Y. Gong, J. Wei, X. Zhang, Y. Xu, Efficiently solving the practical vehicle routing problem: A novel joint learning approach, Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min (2020) 3054–3063, http://dx.doi.org/10.1145/3394486.3403356.

[118] Z. Cao, H. Guo, W. Song, K. Gao, Z. Chen, L. Zhang, X. Zhang, Using reinforcement learning to minimize the probability of delay occurrence in transportation, IEEE Trans. Veh. Technol. 69 (2020) 2424–2436, http://dx.doi.org/10.1109/TVT.2020.2964784.

[119] L. Xin, W. Song, Z. Cao, J. Zhang, Multi-decoder attention model with embedding glimpse for solving vehicle routing problems, in: 35th AAAI Conf. Artif. Intell, 2020, pp. 12042–12049, http://arxiv.org/abs/2012.10638.

[120] J. Zhou, X. Zhou, Multi-echelon inventory optimizations for divergent networks by combining deep reinforcement learning and heuristics improvement, in: Proc. - 2019 12th Int. Symp. Comput. Intell. Des. Isc. 2019, 2019, pp. 69–73, http://dx.doi.org/10.1109/ISCID.2019.00023.

[121] X. Chen, Y. Tian, Learning to perform local rewriting for combinatorial optimization, Adv. Neural Inf. Process. Syst. 32 (2019).

[122] H. Lu, X. Zhang, S. Yang, A learning-based iterative method for solving vehicle routing problems, Iclr 3 (2020) 1–15, https://openreview.net/forum?id=BJe1334YDH.

[123] Y. Wu, W. Song, Z. Cao, J. Zhang, A. Lim, Learning improvement heuristics for solving routing problems, IEEE Trans. Neural Netw. Learn. Syst. (2021) 1–10, http://dx.doi.org/10.1109/TNNLS.2021.3068828.

[124] C. Zhang, W. Song, Z. Cao, J. Zhang, P.S. Tan, C. Xu, Learning to dispatch for job shop scheduling via deep reinforcement learning, in: 34th Conf. Neural Inf. Process. Syst. (NeurIPS 2020), 2020, pp. 1–12, http://arxiv.org/abs/2010.12367.

[125] J. Park, J. Chun, S.H. Kim, Y. Kim, J. Park, Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning, Int. J. Prod. Res. 59 (2021) 3360–3377, http://dx.doi.org/10.1080/00207543.2020.1870013.

[126] Z. Li, Q. Chen, V. Koltun, Combinatorial optimization with graph convolutional networks and guided tree search, Adv. Neural Inf. Process. Syst. 2018 (2018) 539–548.

[127] T. Pierrot, G. Ligner, S. Reed, O. Sigaud, N. Perrin, A. Laterre, D. Kas, K. Beguir, N. de Freitas, Learning compositional neural programs with recursive tree search and planning, Adv. Neural Inf. Process. Syst. 32 (2019).

[128] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A survey of Monte Carlo tree search methods, IEEE Trans. Comput. Intell. AI Games. 4 (2012) 1–43, http://dx.doi.org/10.1109/TCIAIG.2012.2186810.

[129] A. Liu, J. Chen, M. Yu, Y. Zhai, X. Zhou, J. Liu, Watch the unobserved: A simple approach to parallelizing Monte Carlo tree search, ArXiv. 12, pp. 65–73, 2018, http://arxiv.org/abs/1810.11755.

[130] K. Abe, Z. Xu, I. Sato, M. Sugiyama, Solving NP-Hard problems on graphs with extended AlphaGo Zero, 2019, pp. 1–23, 2019, http://arxiv.org/abs/1905.11623.

[131] J. Huang, M. Patwary, G. Diamos, Coloring big graphs with AlphaGoZero, ArXiv, 2019, http://arxiv.org/abs/1902.10162.

[132] A. Laterre, Y. Fu, M.K. Jabri, A.-S. Cohen, D. Kas, K. Hajjar, T.S. Dahl, A. Kerkeni, K. Beguir, Ranked reward: enabling self-play reinforcement learning for combinatorial optimization, ArXiv, 2018, http://arxiv.org/abs/1807.01672.

[133] Z. Xing, S. Tu, L. Xu, Solve traveling salesman problem by Monte Carlo tree search and deep neural network, ArXiv, 2020, http://arxiv.org/abs/2005.06879.

[134] R. Xu, K. Lieberherr, Learning self-play agents for combinatorial optimization problems, Knowl. Eng. Rev. (2020) 2276–2278, http://dx.doi.org/10.1017/S026988892000020X.

[135] Q. Ma, S. Ge, D. He, D. Thaker, I. Drori, Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning, ArXiv, 2019, http://arxiv.org/abs/1911.04936.

[136] P. Emami, S. Ranka, Learning permutations with sinkhorn policy gradient, ArXiv, 2018, http://arxiv.org/abs/1805.07010.

[137] G.A. Malazgirt, O.S. Unsal, A.C. Kestelman, TauRieL: Targeting Traveling salesman problem with a deep reinforcement learning inspired architecture, ArXiv, 2019.

[138] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, ArXiv pp. 1–12, 2017, http://arxiv.org/abs/1707.06347.

[139] Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, A. Cire, Combining reinforcement learning and constraint programming for combinatorial optimization, ArXiv, 1–20, 2020, http://arxiv.org/abs/2006.01610.

[140] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, L.M. Rousseau, Learning heuristics for the tsp by policy gradient, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), in: 10848 LNCS, 2018, pp. 170–181, http://dx.doi.org/10.1007/978-3-319-93031-2_12.

[141] C.K. Joshi, T. Laurent, X. Bresson, An efficient graph convolutional network technique for the travelling salesman problem, ArXiv., pp. 1–17, 2019, http://arxiv.org/abs/1906.01227.

[142] L. Gao, M. Chen, Q. Chen, G. Luo, Z. Liu, N. Zhu, Learn to design the heuristics for vehicle routing problem, ArXiv, pp. 1–10, 2020.

[143] I. Drori, A. Kharkar, W.R. Sickinger, B. Kates, Q. Ma, S. Ge, E. Dolev, B. Dietrich, D.P. Williamson, M. Udell, Learning to solve combinatorial optimization problems on real-world graphs in linear time, in: Proc. - 19th IEEE Int. Conf. Mach. Learn. Appl. ICMLA 2020, 2020, pp. 19–24, http://dx.doi.org/10.1109/ICMLA51294.2020.00013.

[144] J. Song, R. Lanka, Y. Yue, M. Ono, Co-training for policy learning, in: 35th Conf. Uncertain. Artif. Intell. UAI 2019, 2019.

[145] E. Yolcu, B. Póczos, Learning Local Search Heuristics for Boolean Satisfiability, NeurIPS., 2019, pp. 7992–8003.

[146] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, A. Singh, Learning heuristics over large graphs via deep reinforcement learning, in: Assoc. Adv. Artif. Intell., 2019, http://arxiv.org/abs/1903.03332.

[147] D. Selsam, M. Lamm, B. Bünz, P. Liang, D.L. Dill, L. De Moura, Learning a SAT solver from single-bit supervision, in: 7th Int. Conf. Learn. Represent. ICLR 2019, 2019, pp. 1–11.

[148] T. Barrett, W. Clements, J. Foerster, A. Lvovsky, Exploratory combinatorial optimization with reinforcement learning, Proc. AAAI Conf. Artif. Intell. 34 (2020) 3243–3250, http://dx.doi.org/10.1609/aaai.v34i04.5723.

[149] D. Beloborodov, A.E. Ulanov, J.N. Foerster, S. Whiteson, A.I. Lvovsky, Reinforcement learning enhanced quantum-inspired algorithm for combinatorial optimization, Mach. Learn. Sci. Technol. 2 (2021) 025009, http://dx.doi.org/10.1088/2632-2153/abc328.

[150] J. Zhang, H. Zhang, C. Xia, L. Sun, GRAPH-BERT: only attention is needed for learning graph representations, ArXiv, 2020, http://arxiv.org/abs/2001.05140.

[151] Z. Zhang, X. Wang, W. Zhu, Automated machine learning on graphs: A survey, 2021, http://arxiv.org/abs/2103.00742.

[152] D. Mandal, S. Medya, B. Uzzi, C. Aggarwal, Meta-learning with graph neural networks: methods and applications, 2021, http://arxiv.org/abs/2103.00137.

[153] A. Rajeswaran, S.M. Kakade, C. Finn, S. Levine, Meta-learning with implicit gradients, in: Adv. Neural Inf. Process. Syst., vol. 32, 2019, http://arxiv.org/abs/1909.04630.

[154] Y. Zhang, Q. Cai, Z. Yang, Z. Wang, Generative adversarial imitation learning with neural network parameterization: Global optimality and convergence rate, in: 37th Int. Conf. Mach. Learn. ICML 2020. PartF16814, 2020, pp. 10978–10988.

[155] Jonathan. Ho, Stefano. Ermon, Generative adversarial imitation learning, in: 30th Conf. Neural Inf. Process. Syst. (NIPS 2016), 2016, http://dx.doi.org/10.2307/j.ctv1dp0vwx.25.

[156] S. Arora, P. Doshi, A survey of inverse reinforcement learning: Challenges, methods and progress, Artificial Intelligence 297 (2021) 1–48, http://dx.doi.org/10.1016/j.artint.2021.103500.

[157] P. Kamalaruban, R. Devidze, V. Cevher, A. Singla, Interactive teaching algorithms for inverse reinforcement learning, IJCAI Int. Jt. Conf. Artif. Intell. 2019 (2019) 2692–2700, http://dx.doi.org/10.24963/ijcai.2019/374.

[158] Y. Xie, Z. Xu, J. Zhang, Z. Wang, S. Ji, Self-supervised learning of graph neural networks: A unified review, (2021) 1–17, 2021, http://arxiv.org/abs/2102.10757.

[159] Y. Liu, S. Pan, M. Jin, C. Zhou, F. Xia, P.S. Yu, Graph self-supervised learning: A survey, 2021, http://arxiv.org/abs/2103.00111.

[160] J. Devlin, M.W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: NAACL HLT 2019-2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf, vol. 1, 2019, pp. 4171–4186.

[161] X.P. Qiu, T.X. Sun, Y.G. Xu, Y.F. Shao, N. Dai, X.J. Huang, Pre-trained models for natural language processing: A survey, Sci. China Technol. Sci. 63 (2020) 1872–1897, http://dx.doi.org/10.1007/s11431-020-1647-3.

[162] Y. Zhang, Y. Xiong, Y. Ye, T. Liu, W. Wang, Y. Zhu, P.S. Yu, SEAL: Learning heuristics for community detection with generative adversarial networks, in: Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min, 2020, pp. 1103–1113, http://dx.doi.org/10.1145/3394486.3403154.

[163] B. Wilder, E. Ewing, B. Dilkina, M. Tambe, End to end learning and optimization on graphs, Adv. Neural Inf. Process. Syst. 32 (2019) 1–11, http://arxiv.org/abs/1905.13732.

Qi Wang received a B.S. and M.Eng. degree in software engineering from Jilin University (Changchun city) and Central South University (Changsha city) in 2012 and 2016, China, respectively. He is currently pursuing a Ph.D. degree in software engineering at the school of computer science, Fudan University, Shanghai, China.

His current research interests include combinatorial optimization, deep learning, and reinforcement learning.

Chunlei Tang (corresponding author) is a research associate at Harvard Medical School. Dr. Tang, an author of The Data Industry: The Business and Economics of Information and Big Data (Wiley; 2016) and Data Capital: How Data is Reinventing Capital for Globalization (Springer; 2021), is a recognized advocate for the Data Economy. She is a member of the ACM-W (ACM Women) leadership team as Treasurer Co-Chair. Tang also serves as the Founder President of the Harvard Data Entrepreneurship Club. Tang's work begins with an innovative data-mining method built upon her doctoral work and applicable to various fields (e.g., finance, transportation, economics, insurance, bioinformatics, and sociology). She then endeavored to better match data products with their marketing capabilities by tackling business innovation. Inspired by business, she implemented a theoretical innovation for Data Science and proposed the (now-mainstream) definition of the data industry in 2013, stating that the data industry aims to bridge the gap between data science and economics. Her current focus is on applying innovation in healthcare, and she believes precision medicine is an application of the data industry.