Front. Comput. Sci., 2025, 19(6): 196322
https://doi.org/10.1007/s11704-024-40490-y

REVIEW ARTICLE

A survey on deep learning-based algorithms for the traveling salesman problem

Jingyan SUI[1,2], Shizhe DING[1,2], Xulin HUANG[1,4], Yue YU[1,2,5], Ruizhi LIU[1,2], Boyang XIA[1,2], Zhenxin DING[1,2], Liming XU[1,2], Haicang ZHANG[1,2], Chungong YU[1,2], Dongbo BU (✉)[1,2,3]

1 State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
2 University of Chinese Academy of Sciences, Beijing 100190, China
3 Central China Institute of Artificial Intelligence, Zhengzhou 450046, China
4 Henan Institute of Advanced Technology, Zhengzhou University, Zhengzhou 450002, China
5 Hangzhou Institute for Advanced Study, UCAS, Hangzhou 310024, China

Abstract This paper presents an overview of deep learning (DL)-based algorithms designed for solving the traveling salesman problem (TSP), categorizing them into four categories: end-to-end construction algorithms, end-to-end improvement algorithms, direct hybrid algorithms, and large language model (LLM)-based hybrid algorithms. We introduce the principles and methodologies of these algorithms, outlining their strengths and limitations through experimental comparisons. End-to-end construction algorithms employ neural networks to generate solutions from scratch, demonstrating rapid solving speed but often yielding subpar solutions. Conversely, end-to-end improvement algorithms iteratively refine initial solutions, achieving higher-quality outcomes but necessitating longer computation times. Direct hybrid algorithms directly integrate deep learning with heuristic algorithms, showcasing robust solving performance and generalization capability. LLM-based hybrid algorithms leverage LLMs to autonomously generate and refine heuristics, showing promising performance despite being in early developmental stages. In the future, further integration of deep learning techniques, particularly LLMs, with heuristic algorithms and advancements in interpretability and generalization will be pivotal trends in TSP algorithm design. These endeavors aim to tackle larger and more complex real-world instances while enhancing algorithm reliability and practicality. This paper offers insights into the evolving landscape of DL-based TSP solving algorithms and provides a perspective for future research directions.

Keywords traveling salesman problem, algorithms design, deep learning, neural network

1 Introduction

The traveling salesman problem (TSP) is defined as follows: given a series of cities, the objective is to determine the shortest tour that starts from any city, visits each city exactly once, and ultimately returns to the starting city. Due to its NP-hard essence, solving TSP is exceedingly challenging. However, despite its complexity, numerous researchers persist in developing algorithms to address it.

This sustained interest is rooted in the critical role that TSP plays not only in the domain of theoretical computer but also in operations research and mathematical optimization [1]. Furthermore, TSP holds substantial practical significance, as it forms the basis for addressing a multitude of real-world challenges. For example, tasks like package delivery routing, printed circuit drilling, and order picking in warehouses can all be conceptualized as TSP instances [2].

In recent years, deep learning (DL) has brought revolutionary progress to many fields [3]. For instance, DL has facilitated scientists in diagnosing cancer [4], predicting infectious diseases [5], discovering physical laws [6–8], predicting protein structures [9–12], proving mathematical theorems [13], performing matrix multiplication [14], conducting symbolic regression [15], discovering mathematical algorithms [16], and mining spatio-temporal data [17–23]. These examples highlight the transformative potential of DL across various disciplines.

DL techniques offer a new "data-driven" approach to problem-solving. By harnessing neural network models, these techniques acquire data representations at multiple hierarchical levels, adeptly capturing intricate patterns and underlying regularities within the data [3], thereby facilitating researchers in addressing problems.

DL has also been employed in designing algorithms for TSP, offering new insights for researchers. Traditional TSP algorithm design typically relies heavily on manual intervention by researchers, who craft algorithms based on a limited number of instances. This approach is time-consuming and labor-intensive. In contrast, DL can inspire researchers and assist them in efficiently designing algorithms.

Received May 14, 2024; accepted June 20, 2024

E-mail: dbu@ict.ac.cn

Furthermore, DL, adept at modeling intricate functions, has the potential to supplant cumbersome computations in traditional algorithms. This replacement not only expedites the computational process but also furnishes supplementary information or decision criteria for traditional algorithms [24].

Additionally, the rise of large language model (LLM) has introduced new possibilities for TSP algorithm design. LLMs can leverage vast amounts of data and learned knowledge to automatically generate and refine heuristics. This capability allows for the exploration of a broader heuristic space without being constrained by human-designed components. By integrating LLMs into the algorithm design process, researchers can automate the generation of novel heuristics, optimize them iteratively, and potentially discover more effective TSP solutions. This shift towards automated heuristic design represents a significant advancement, reducing dependency on manual efforts and enhancing the overall efficiency and innovation in TSP algorithm development.

With the advancement of DL, a series of novel algorithms for TSP have emerged. To our knowledge, over the past three years, three surveys [25–27] have explored different facets of DL-based algorithms for TSP. However, they have covered relatively fewer algorithms and primarily focused on theoretical analysis, lacking experimental validation and discussion. These surveys are briefly described as follows:

(1) Mele et al. [25] provided an intuitive mind map for navigating the existing literature on DL-based applications in combinatorial optimization, with a particular focus on the Traveling Salesman Problem (TSP). They classified algorithms into auto-regressive and one-shot flows, explored various machine learning techniques for solving TSP, and highlighted open research challenges in the field.

(2) Shi et al. [26] reviewed neural network methods for TSP solving, categorizing them into Hopfield neural network, graph neural network, and neural network with reinforcement learning.

(3) Yang et al. [27] conducted a review of reinforcement learning (RL) in combinatorial optimization, comparing historical and modern RL algorithms for TSP solving. They introduced deep reinforcement learning (DRL), elaborating on its departure from traditional mathematical frameworks and its integration with attention and feature encoding mechanisms.

In comparison to these surveys, ours presents a more comprehensive analysis. We offer detailed categorization of algorithms based on technological distinctions and perform experimental validation and analysis on representative algorithms from each category, employing both randomly generated and real-world TSP instances. Moreover, we assess their generalization performance, thereby enhancing the comprehension of DL's applicability in algorithm design for TSP.

Besides, none of the three surveys delved into the application of LLM technology in algorithm design for TSP. However, this could be attributed to the possibility that LLMs were not yet applied in this field at the time. In contrast, ours introduces the application of LLMs in algorithm design for TSP, addressing this gap.

In summary, this paper provides a comprehensive summary and comparison of DL-based algorithms for TSP, categorizing them into two main categories: i) DL-based end-to-end algorithms, ii) Hybrid algorithms combining DL with heuristic algorithms.

DL-based end-to-end algorithms can be further divided into two sub-categories: i) End-to-end construction algorithms, and ii) End-to-end improvement algorithms.

Hybrid algorithms can also be divided into two sub-categories: i) Direct hybrid algorithms, and ii) LLM-based hybrid algorithms.

The structure of this paper is as follows: Section 1 provides an introduction to the background and motivation of incorporating DL into solving TSP; Section 2 provides the definition of TSP and challenge of solving it; Section 3 briefly introduces traditional algorithms; Section 4 reviews DL-based algorithms for TSP, categorizing them and analyzing their principles, advantages, drawbacks, and performance; Section 5 evaluates performance of representative algorithms and discusses the findings; Section 6 concludes the paper and outlines future research directions.

2 Definition and challenge of TSP

2.1 Definition of TSP

The definition of TSP is as follows: given a node set $V = \{v_1, \ldots, v_n\}$, an edge set $E = \{(r, s) : r, s \in V\}$, and a cost metric d_{rs} for each edge $(r, s) \in E$, the objective of the TSP is to find the minimum-cost tour that visits each node exactly once [28].

TSP encompasses various types, broadly classified as the symmetric traveling salesman problem (sTSP), asymmetric traveling salesman problem (aTSP), and multi-traveling salesman problem (mTSP) [28]. The problem is symmetric if $d_{rs} = d_{sr}$ for all $(r, s) \in E$, otherwise, it is asymmetric. The mTSP involves multiple salesmen departing from a depot to visit other cities before returning to the depot, with each city visited only once, optimizing the total cost.

A subclass of sTSP, defined in a plane with Euclidean distance as the cost metric, is known as the euclidean symmetric traveling salesman problem. DL-based algorithms primarily target this subclass, which will be henceforth referred to simply as the TSP.

Figure 1 represents a TSP instance with 50 nodes, denoted as TSP50. In the figure, blue nodes represent cities, gray lines denote paths connecting cities, and the red loop denotes the optimal solution.

2.2 Challenge of solving TSP

TSP poses significant computational challenges. Algorithms such as exhaustive search, branch and bound [29,30], and dynamic programming [31] suffice for small-scale instances. However, with expanding problem sizes, these methods necessitate substantial computational resources, hindering scalability to larger instances.

For a TSP with n nodes, there exist $(n-1)!/2$ possible tours [28]. As n grows, feasible tour permutations increase

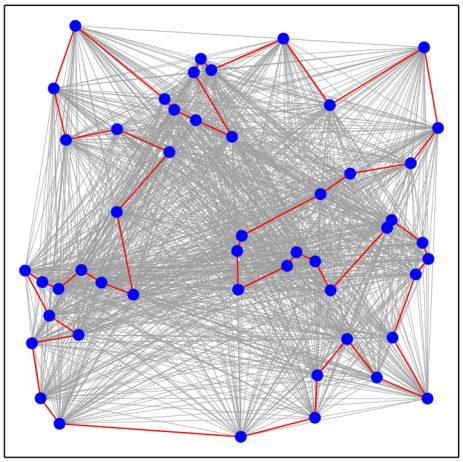

Fig. 1 A TSP instance with 50 nodes (TSP50)

exponentially. For example, with $n = 20$, there are approximately 10^{16} tours; with $n = 50$, about 10^{62}; and with $n = 100$, about 10^{155}. Hence, pinpointing the shortest tour amidst such vast solution spaces poses a formidable challenge.

3 Traditional algorithms for TSP

Traditional algorithms mainly consist of exact algorithms and heuristic algorithms [32].

3.1 Exact algorithms

Exact algorithms utilize explicit or implicit enumeration techniques to search the entire solution space, aiming to find the global optimum. Common approaches include modeling TSP as dynamic programming [31] or integer linear programming [33,34]. Integer linear programming is then solved using branch and bound [29,30] or branch and cut [35]. Concorde [36] is widely recognized as a premier exact solver, known for its efficiency and precision in solving TSP instances. Its optimization algorithms are adept at handling diverse TSP variants, establishing it as a dependable benchmark for assessing the effectiveness of DL-based algorithms in combinatorial optimization tasks, particularly in TSP.

Although exact algorithms can obtain optimal solutions for TSP, they are not polynomial-time algorithms and inevitably consume a considerable amount of time, making them impractical for solving large-scale TSP instances.

3.2 Heuristic algorithms

Unlike exact algorithms, which always find the optimal solution, heuristic algorithms aim to find sufficiently good approximate solutions. Heuristic algorithms encompass nearest neighbor algorithm [37], insertion algorithm [37], and local search algorithms [35,38,39], among others. This type of algorithms also encompasses metaheuristic algorithms inspired by physical and biological principles, such as evolutionary algorithms including genetic algorithms, ant colony algorithm [1].

Local search algorithm is commonly used heuristics that iteratively explores solutions within neighborhood of a current solution. Specifically, local search algorithms typically employ improvement heuristics, such as k-OPT [35] and relocation [38], to improve a feasible solution iteratively until a higher-quality approximate solution is obtained. The k-OPT involves removing k inferior edges from a feasible solution and adding k superior edges to reconnect the remaining segments of the solution [35]. The relocation relocates a node from its current position to a new position within the solution [38]. Local search algorithms typically iterate from any feasible solution, often converging to a local optimum rather than a global optimum [39].

In k-OPT heuristics, the most widely used techniques are 2-OPT and 3-OPT [40], as depicted in Fig. 2. The lines in the figure are merely schematic representations of routes, and their lengths do not represent the actual lengths of the routes.

In 2-OPT, two inferior edges (represented by the blue lines in the 2-OPT section of Fig. 2) are removed from the current tour, while two superior edges (illustrated by the red lines in the 2-OPT segment of Fig. 2) are added to reconnect the remaining segments, resulting in a shorter tour. Similarly, the 3-OPT entails the removal of three inferior edges and the addition of three superior edges.

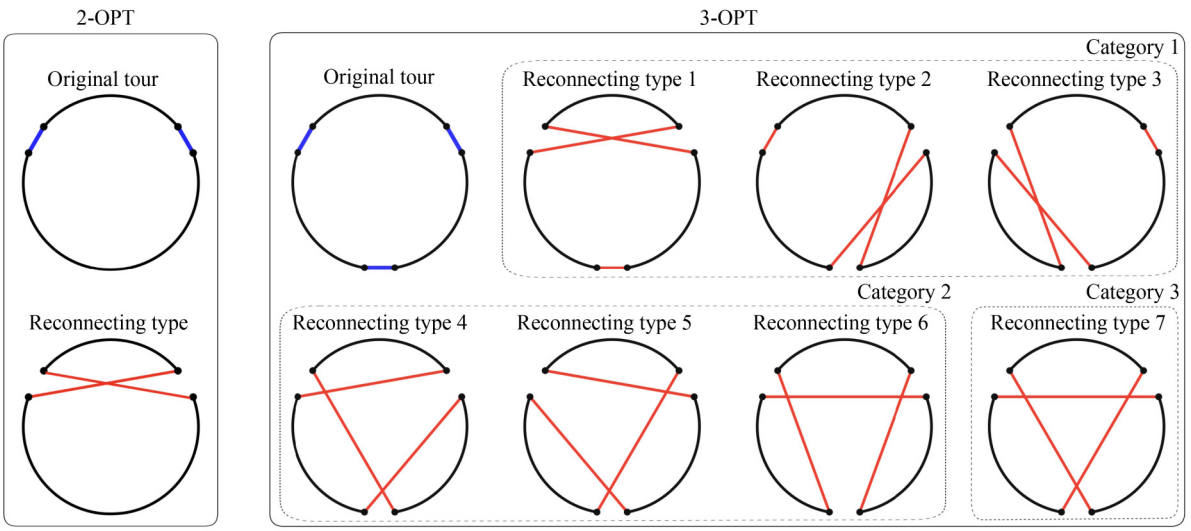

Fig. 2 2-OPT and 3-OPT heuristics for TSP [41]

The search capability of 3-OPT surpasses that of 2-OPT, increasing the likelihood of discovering superior solutions. However, the implementation complexity of 3-OPT outweighs that of 2-OPT [38,42]. The primary difference between 3-OPT and 2-OPT lies in the difficulty of reconnecting remaining segments. 3-OPT presents a higher reconnection difficulty compared to 2-OPT, because 3-OPT offers up to seven different reconnection types [41], whereas 2-OPT only has one reconnection type.

The Lin-Kernighan-Helsgaun (LKH) algorithm [43–45] (LKH, LKH-2, LKH-3) are typical heuristic algorithms capable of effectively solving routing problems, including TSP.

Heuristic algorithms typically find high-quality approximate solutions quickly. However, designing heuristic algorithms is a challenging task as it heavily relies on manually crafting appropriate heuristics. This manual design process depends excessively on the intuition and experience of researchers, consuming significant time and effort.

4 Research status analysis

4.1 Overview of DL-based algorithms for TSP

In recent years, numerous DL-based algorithms for TSP have emerged [41,46–75]. According to whether the algorithm framework solely relies on DL, we categorize these algorithms into two main categories: i) DL-based end-to-end algorithms, ii) Hybrid algorithms combining DL with heuristics.

4.1.1 DL-based end-to-end algorithms

DL-based end-to-end algorithms employ data-driven strategies, enabling neural networks to directly learn heuristics from a large number of TSP instances without any manual intervention. After training the neural network, TSP instances are inputted into the neural network from the “input end”, and solutions are generated based on learned heuristics and directly outputted from the “output end”. This is why this type of algorithms is called end-to-end algorithms.

Based on the way solutions are generated, we further divide DL-based end-to-end algorithms into two sub-categories: construction algorithms and improvement algorithms.

- DL-based end-to-end construction algorithms generally utilize neural networks to acquire construction heuristics. Once learned, these heuristics are commonly employed in an auto-regressive manner (with some exceptions) to construct complete solutions [46–61]. Auto-regressive manner implies selecting the next node based on the nodes already chosen, gradually building the complete solution from scratch.
- DL-based end-to-end improvement algorithms typically utilize neural networks to learn improvement heuristics. These heuristics, once learned, are used to iteratively improve initial solutions (which are randomly generated or generated using simple heuristics) to obtain high-quality solutions [41,62–64].

4.1.2 Hybrid algorithms combining DL with heuristics

Hybrid algorithms are a method that combines DL with heuristic algorithms. Depending on the way they are combined, they can be divided into two sub-categories: direct hybrid algorithms and LLM-based hybrid algorithms.

- Direct hybrid algorithms integrates DL directly into the search framework of heuristic algorithms, combining the powerful learning ability of DL with the excellent search capability of heuristic algorithms to search for solutions in the solution space [65–72]. These algorithms utilize DL to learn features and patterns of TSP, and then use the acquired knowledge to guide search process.
- LLM-based hybrid algorithms utilize large language modes to assist in the automatic generation and evolution of heuristics [73–75]. Specifically, these algorithms utilize large language modes to generate new heuristics and guide evolution process of heuristics.

In the following subsections, we will firstly present the key techniques of DL applied to TSP. Subsequently, we will delve into typical DL-based algorithms. The classification of DL-based algorithms for TSP is depicted in Fig. 3.

4.2 DL techniques for TSP

DL employs deep neural networks as computational models to mine latent patterns and complex structures in large-scale datasets [3]. Currently, various DL techniques and neural networks have been developed. Some of these techniques have been introduced for solving TSP and have demonstrated promising results.

In this section, we briefly outline the learning paradigms and neural network architectures commonly employed for TSP. The learning paradigms encompass supervised learning (SL) [24] and reinforcement learning (RL) [24], while the neural network architectures include attention mechanisms [76], graph neural network (GNN) [77], Transformer [78], and large language models [79].

4.2.1 Learning paradigm

The most common paradigm of learning for TSP mainly include supervised learning and reinforcement learning.

In supervised learning, the model learns mapping relationship between input and output by training on labeled data [24].

In contrast, reinforcement learning, without the need for pre-generated data labels, involves the interaction between an agent and its environment through Markov decision processes (MDP), as shown in Fig. 4. At each time step, the agent selects actions based on the environment’s state and adjusts its policy based on the feedback of received rewards to maximize cumulative rewards.

For supervised learning, labeling datasets can be a resource-intensive process, particularly when tackling problems like TSP with its exponentially increasing complexity. As the problem scale increases, obtaining precise solutions through exact algorithms as labels becomes challenging. Using approximate algorithms may yield less accurate labels, resulting in unfavorable training outcomes.

Hence, reinforcement learning, which circumvents the need for data labeling, has been applied for addressing TSP.

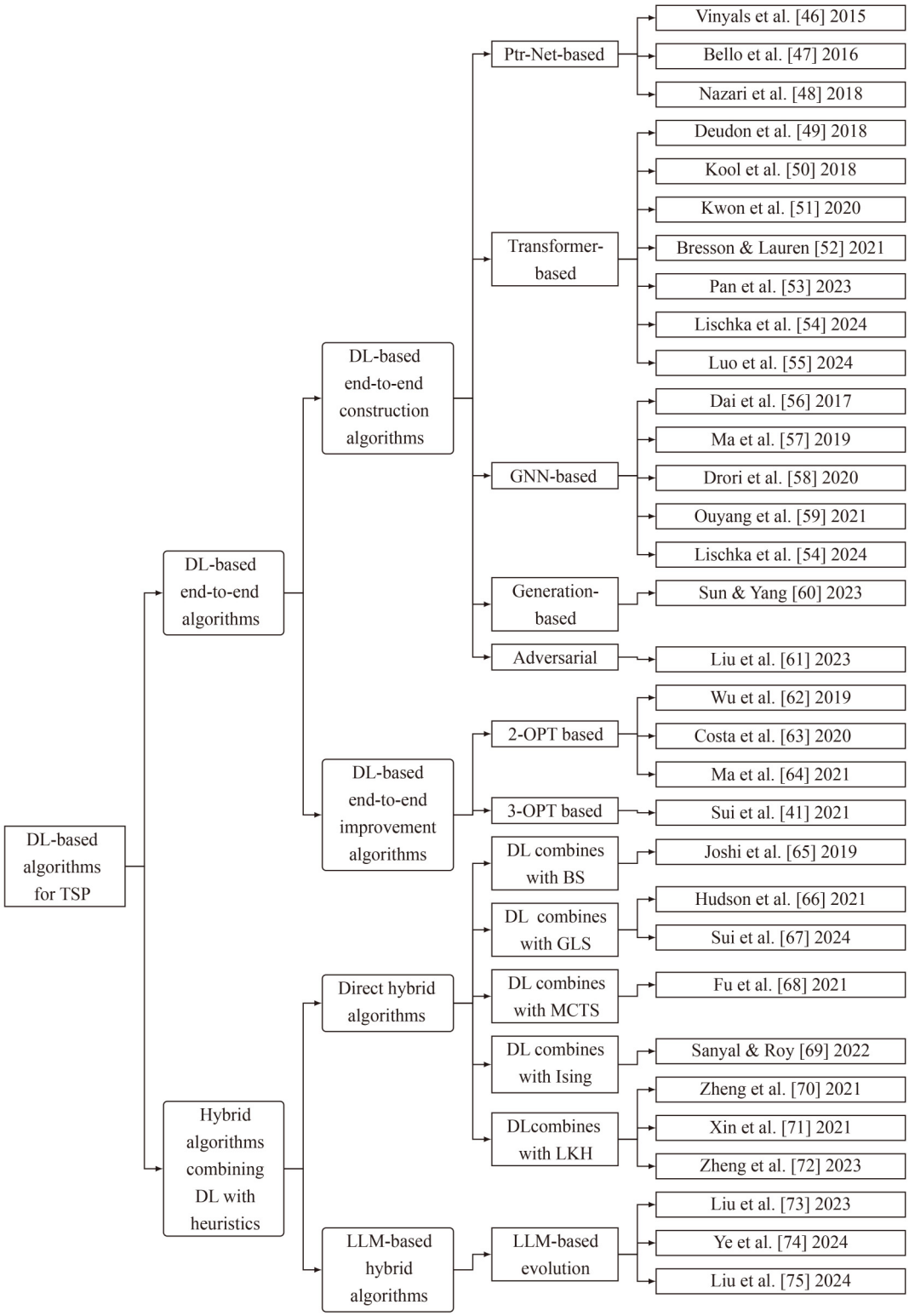

Fig. 3 Classification of DL-based algorithms for TSP

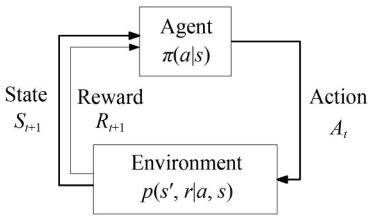

Fig. 4 Markov decision process associated with RL [24]

4.2.2 Neural network architecture

● Attention mechanism

In 2014, Bahdanau et al. [76] proposed the attention mechanism for machine translation. It enables a better alignment of semantic relationships between the source and target languages, thus enhancing translation accuracy and fluency. Subsequently, researchers have proposed various

forms of attention mechanisms, such as multidimensional attention, hierarchical attention, self-attention, memory-based attention, and task-specific attention [80]. In essence, the attention mechanism assists neural network models in identifying the crucial parts from complex information.

The formula for attention mechanism is given by [76]:

$$\begin{aligned} e_{tj} &= a(s_{t-1}, h_j), \\ \alpha_{tj} &= \frac{\exp\left(e_{tj}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{tk}\right)}, \\ c_t &= \sum_{j=1}^{T_x} \alpha_{tj} h_j. \end{aligned} \tag{1}$$

In Eq. (1), $a(s_{t-1}, h_j)$ is the alignment function used to compute attention scores; s_{t-1} can be viewed as a vector specific to a particular task, used to match with each element in the sequence $\{h_j\}$ and output attention scores e_{tj} to measure the degree of matching between s_{t-1} and h_j. The attention scores e_{tj} are normalized to obtain attention weights α_{tj}, based on which the context vector c_t is computed for subsequent calculations.

Based on different alignment function computation methods, there are two fundamental forms of attention mechanisms: additive attention proposed by Bahdanau et al. [76], and dot-product attention proposed by Luong et al. [81].

As depicted in Fig. 5, attention mechanisms empower models to dynamically concentrate on various segments of the input data, leading to more effective outcomes. In the case of TSP, the impact of distinct neighboring nodes on a given node fluctuates. Thus, when determining the next node to visit, it becomes imperative to allocate different weights to individual neighbors. Attention mechanisms adeptly fulfill this necessity in TSP.

● Transformer

In 2017, Vaswani et al. [78] introduced a novel neural architecture known as the Transformer model, which garnered significant attention in the field of natural language processing. Transformer relies entirely on the attention mechanism [81] and eschews the prevalent use of recurrent or convolutional networks at the time.

The Transformer model adopts an encoder-decoder structure, as illustrated in Fig. 6.

The encoder consists of N identical layers, each composed of two sub-layers: a multi-head self-attention mechanism (MHA) and a fully connected feed-forward network (FF). Each sub-layer incorporates residual connections and layer normalization operations.

The decoder comprises N identical layers, although with slight differences in the structure of each layer compared to the encoder. In addition to the multi-head self-attention mechanism and the fully connected feed-forward network, each decoder layer includes a masked multi-head attention sub-layer. Residual connections and layer normalization operations are applied to each sub-layer.

The model's input consists of input data and positional encoding, while the output undergoes normalization followed by the softmax function to obtain a probability distribution.

Experimental results demonstrate that Transformer outperforms previous models in terms of quality, with higher parallelization, reduced training time, and strong generalization to other tasks. Based on these advancements, the Transformer model has also been applied to addressing TSP.

● Graph neural networks

Graph neural networks (GNNs) refer to a class of neural network models applied to graphs, suitable for various graph-related problems [77]. As early as 1997, Sperduti and Starita [82], as well as Baskin et al. [83], first utilized neural networks to extract features from graph data instead of

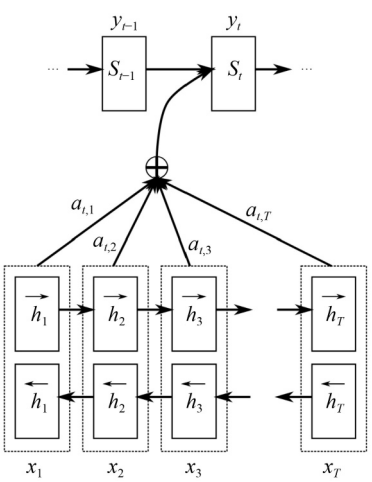

Fig. 5 Attention mechanism [76]

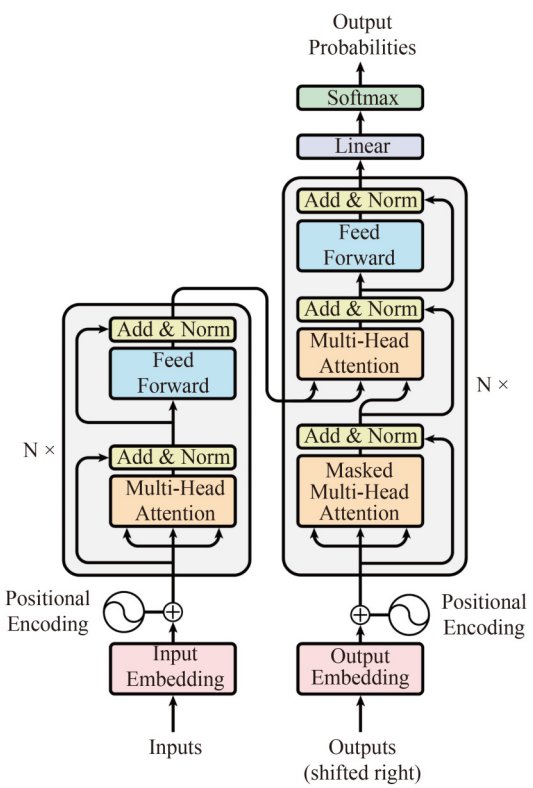

Fig. 6 Transformer [78]

manually designing features. Subsequently, Gori et al. [84] in 2005 and Scarselli et al. [85] in 2008 proposed novel graph learning models named Graph Neural Networks. Researchers have since introduced a series of variations of GNNs [86–93]. Among these, graph convolutional network (GCN) and graph attention network (GAT) are particularly noteworthy.

Graph convolutional networks employ convolution operations in graph neural networks and have evolved into various versions. For instance, in 2017, Bresson and Laurent [92] introduced a residual gated graph convolutional network, as illustrated in Fig. 7. This network adds residual connections and gating mechanisms to the conventional graph convolutional network, enabling the network to utilize more layers and aggregate information from current nodes and neighboring nodes more flexibly to generate node embeddings.

The computation formula for residual gated graph convolutional networks is [92]:

$$h_i^{\ell+1} = f^{\ell}\left(h_i^{\ell}, \left\{h_j^{\ell} : j \to i\right\}\right) + h_i^{\ell}, \tag{2}$$

where $h_i^{\ell+1}$ represents the embedding of node i in layer $\ell+1$, h_i^{ℓ} represents the embedding of node i in layer ℓ, h_j^{ℓ} represents the embedding of node j in layer ℓ, and $j \to i$ indicates that node j is a neighbor node of node i.

In 2017, Veličković et al. [93] proposed graph attention network (GAT), as depicted in Fig. 8. This novel neural network architecture, based on attention mechanisms, effectively handles graph-structured data. It computes different weights for nodes in the neighborhood without employing any expensive matrix operations (such as inversion) or prior knowledge of the graph structure. This enables nodes to effectively aggregate features of their neighboring nodes, thereby enhancing the effectiveness of graph data representation learning.

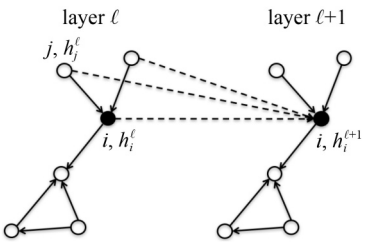

Fig. 7 Graph convolutional network [92]

● Large language model

Large language models (LLMs) refer to DL models with billions to tens of billions of parameters, exemplified by the GPT series [79]. They are pre-trained on massive text corpora and fine-tuned for various natural language processing tasks, showcasing strong performance in tasks such as text generation, machine translation, sentiment analysis, and more [94].

The large models have exhibited outstanding performance not only in the field of natural language processing [95], but also in a variety of research areas, including programming [96], chemistry [97], medicine [98–100], chip design [101,102].

With the enhancement of capabilities in LLMs, they have recently been applied in the field of algorithm design, leading to a series of successes [103–107].

4.3 DL-based end-to-end construction algorithms

DL-based end-to-end construction algorithms typically utilize neural networks to learn constructive policies and autonomously generate a complete solution step-by-step from scratch.

These algorithms [46–61] generally employ an encoder-decoder architecture. The encoder generates embeddings for nodes, and the decoder computes the probability distribution of candidate nodes at each decoding step based on the embeddings of candidate nodes and selected nodes in previous steps. A node is selected according to this probability distribution and added to the current path. The decoding process iterates until a cycle containing all nodes is constructed.

While DL-based end-to-end construction algorithms are highly efficient, the solutions they produce are frequently deemed inadequate in quality.

In end-to-end construction algorithms, both supervised learning and reinforcement learning have been employed to train neural networks. Since reinforcement learning does not require pre-solved solutions as training labels but only

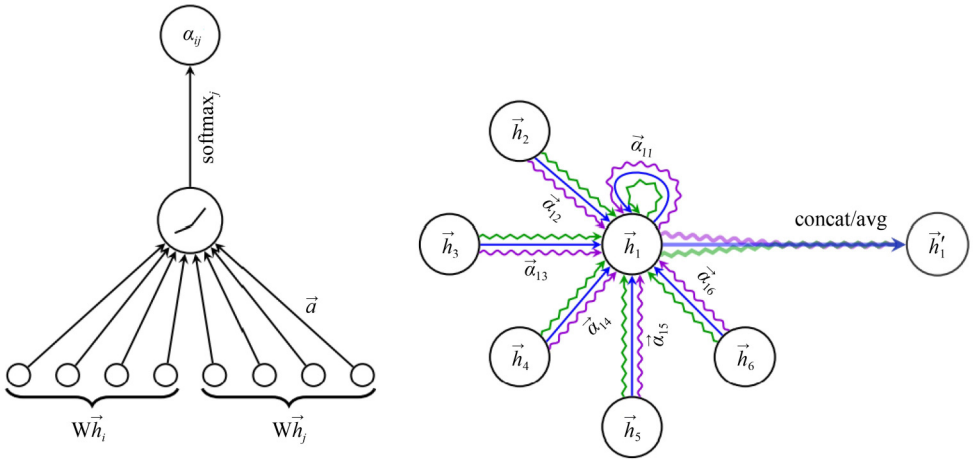

Fig. 8 Graph attention network [93]

sufficient TSP instances, it has gradually become the primary learning paradigm for construction algorithms.

DL-based end-to-end construction algorithms primarily consist of algorithms based on pointer network, transformer, and graph neural network. Additionally, they incorporate generation-based and adversarial algorithms.

4.3.1 Pointer network-based algorithms

● Pointer network

As early as 2015, Vinyals et al. [46] proposed pointer network (Ptr-Net) for solving TSP, initiating a series of research endeavors utilizing DL for TSP. The work treats TSP as sequential data, drawing from the Sequence-to-Sequence (Seq2Seq) [108] model in the field of machine translation, and integrates attention mechanism into the design of their model.

The significance of this work lies in successfully integrating deep neural networks with TSP. While Seq2Seq models are a prevalent method for handling sequence data in translation tasks, their fixed output dimensions posed challenges when applied to TSP, where the output probability distribution dimension varies with problem scale. Ptr-Net overcame this limitation by introducing a simplified variant of the attention mechanism within the Seq2Seq model framework, allowing model to output sequences with dimensions consistent with the input sequence, thus providing a DL-based algorithm for TSP.

Specifically, Ptr-Net incorporated a simplified attention mechanism into the Seq2Seq model, where attention weights are directly treated as the probability distribution of nodes in the input sequence, akin to a "pointer" directly indicating nodes in the input sequence, as illustrated in Fig. 9. This design inspired the name "Pointer network".

The attention mechanism formula for the Ptr-Net is as follows [46]:

$$\begin{aligned} u_j^i &= v^T \tanh\left(W_1 e_j + W_2 d_i\right), \quad j \in (1,\ldots,n), \\ p(C_i \mid C_1,\ldots,C_{i-1},\mathcal{P}) &= \mathrm{softmax}\left(u^i\right), \end{aligned} \tag{3}$$

where e_j and d_i represent the hidden vectors of Ptr-Net's encoder and decoder, respectively; $\mathcal{P}$ denotes coordinates of nodes in the input sequence $\{[x_1,y_1],[x_2,y_2],\ldots,[x_n,y_n]\}$; C_i represents the ith node in the output sequence; and v, W_1, and W_2 are learnable parameters.

Ptr-Net employs an encoder-decoder structure based recurrent neural network (RNN) [109]. Specifically, both the encoder and decoder utilize a specialized type of RNN known as long short-term memory (LSTM) network [110]. Moreover, a simplified variant of the attention mechanism is utilized in the decoder to compute the probability distribution of nodes in the input sequence. At each decoding step, a node is selected based on the computed probability distribution and appended to the current sequence until a complete loop containing all nodes is formed.

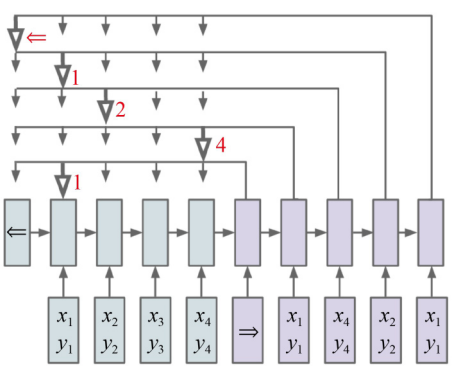

Fig. 9 Pointer network [46]

Due to the model's tendency to output infeasible solutions, beam search is employed to filter out such solutions.

This algorithm employs supervised learning to train model, enabling it to rapidly solve new instances with distributions similar to the training dataset. However, supervised learning necessitates a large amount of training data with optimal or high-quality solutions as labels, which requires significant computational cost and time to construct, thereby limiting the scalability of the algorithm.

Ptr-Net was tested on TSP instances ranging from 5 to 50 nodes (referred to as TSP5 and TSP50). Ptr-Net represents an early stage in the development of this category of algorithms. Despite addressing small-scale instances with solutions of less-than-ideal quality, it has paved the way for subsequent research in new directions.

● NCO framework

Due to the need for optimal or high-quality solutions as training labels in supervised learning, which entails significant computational costs and time, in 2016, Bello et al. [47] proposed a neural combinatorial optimization (NCO) framework for solving TSP using reinforcement learning. Unlike supervised learning, reinforcement learning does not require labeled training data; instead, TSP instances serve as training samples, and neural network learns by interacting with the environment and receiving rewards. For TSP, the expected tour length typically serves as rewards.

In NCO framework, Bello et al. [47] employed a Ptr-Net as the policy model for reinforcement learning and improved its attention mechanism to ensure the model consistently generates feasible solutions. During each decoding step, the model first masks off the nodes already visited in the attention vector (by setting their values to negative infinity), and then normalizes the attention vector using the softmax function to obtain the probability distribution over nodes. This ensures that the pointer always points to unselected nodes during decoding, thereby yielding feasible solution.

The NCO framework employs policy gradient algorithm from reinforcement learning, specifically the actor-critic algorithm [111], to train policy model.

Additionally, this algorithm employs two search strategies, sampling and active search, to further improve the quality of solutions during inference. The sampling strategy involves sampling multiple candidate solutions and selecting the shortest one among them, while the active search strategy improves the model's parameters while reasoning about each test instance.

NCO framework was tested on TSP20, TSP50, and TSP100 datasets. It was found that pretraining with reinforcement learning followed by active search yielded the best results. The solution quality of NCO framework surpassed that of

Vinyals et al.'s Ptr-Net [46] trained with supervised learning, as well as the Christofides algorithm [112] and Google's open-source solver OR-Tools [113].

● Attention + RL algorithm

In 2018, Nazari et al. [48] simplified Ptr-Net by removing the RNN in the encoder and retaining only the RNN in the decoder. As depicted in Fig. 10, they directly used embeddings of static information (such as node coordinates) as input to the decoder. They then computed embeddings of both static and dynamic information (like node demands) in the decoder, generating node distribution with an attention mechanism. This simplified Ptr-Net effectively computes embeddings of node feature after dynamic changes and ensures the model is unaffected by the input sequence order of nodes.

This algorithm still employs policy gradient algorithm from reinforcement learning for training within the NCO framework.

It can solve both TSP and vehicle routing problem (VRP) with dynamic features.

The algorithm was also tested on TSP20, TSP50, and TSP100 datasets, with results indicating comparable solution quality to Bello et al.'s algorithm [47], while reducing training time by approximately 60%.

4.3.2 Transformer-based algorithms

Due to Transformer's high quality, parallelization capabilities, and strong generalization performance, It was applied to solving TSP.

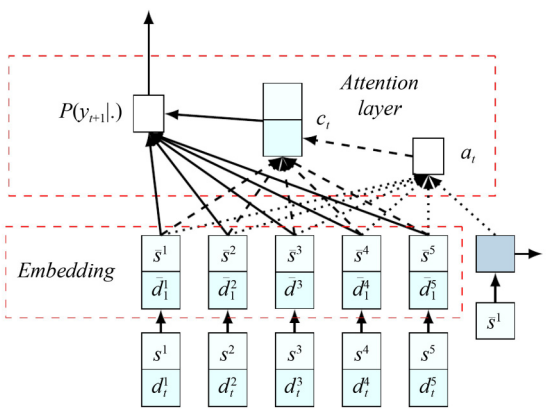

Fig. 10 Attention+RL algorithm [48]

● Multi-head Attention + RL algorithm

In 2018, Deudon et al. [49] utilized Transformer to propose their algorithm. Their model adopts an encoder-decoder architecture.

The encoder employs Transformer's multi-head attention mechanism and feed forward layers. They eliminated positional encodings to ensure that the node representations obtained are independent of the input sequence order. Their input comprises all city coordinates, while the output is the embeddings for each node, also referred to as the action vector.

The decoder employs a pointer mechanism akin to the Ptr-Net. However, instead of using all previously selected nodes to generate a query vector, they only utilize the last three sampled nodes to predict the next selected node, thus avoiding the use of LSTM. They apply the same masking mechanism as Bello et al. [47] to mask visited nodes and ensure the constructed tour is feasible.

This algorithm also utilizes policy gradient algorithm for model training.

Additionally, they preprocess the input data using principal component analysis (PCA) to achieve rotation invariance in the input space. Moreover, the algorithm integrates 2-OPT algorithm to further enhance solution quality.

The solution quality of this algorithm on TSP20 and TSP50 surpasses that of Christofides algorithm [112] and Ptr-Net [46], comparable to NCO framework [47] and OR-Tools [113]. The model trained on TSP50 data also performs well on TSP100 instances. When combined with 2-OPT algorithm, there is a significant improvement in solution quality.

● AM algorithm

In 2018, Kool et al. [50] also drew inspiration from Transformer to propose their model, named attention model (AM), which adopts an encoder-decoder architecture.

The encoder, illustrated in Fig. 11, employs Transformer's multi-head attention mechanism and feed forward layers, while eliminating positional encodings. It takes all node coordinates as input and produces embeddings for each node, with the mean of node embeddings representing the graph embedding of TSP.

The decoder, depicted in Fig. 12, employs a single attention

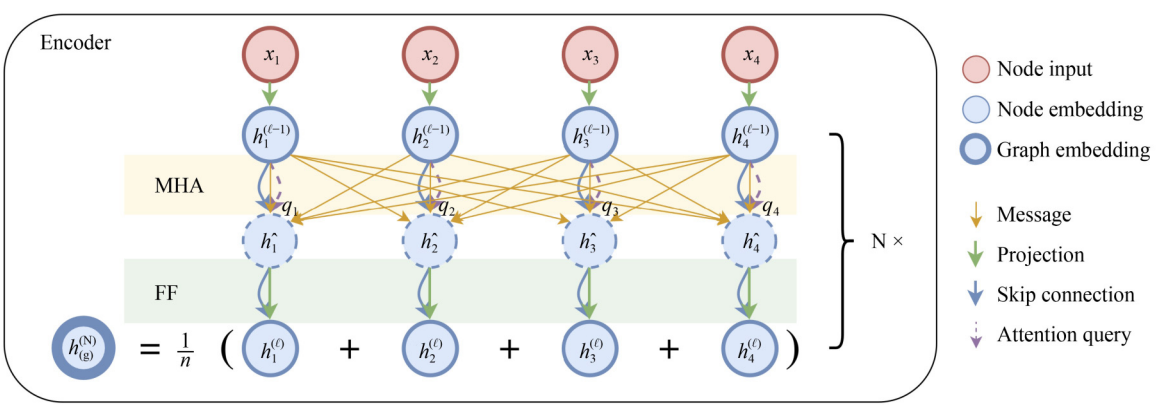

Fig. 11 Encoder of AM algorithm [50]

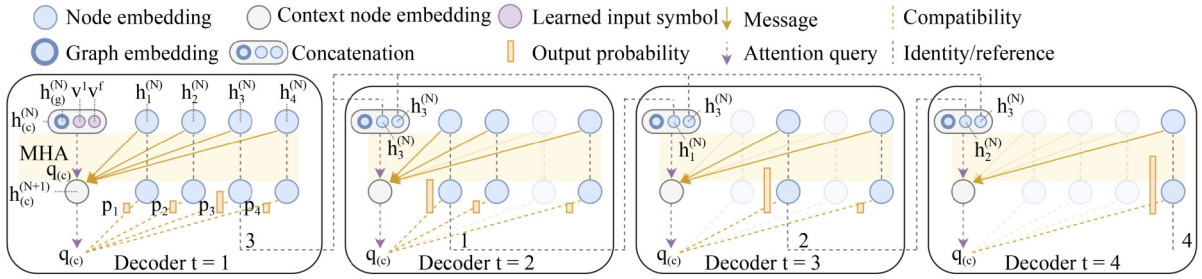

Fig. 12 Decoder of AM algorithm [50]

head. At each decoding step, the current output is derived based on the embeddings from the encoder and the previous output. Specifically, it constructs a context vector to compute the query vector, incorporating the embeddings of TSP graph, the node selected in previous step, and the initial node. Subsequently, the query vector and node embeddings are fed into decoder to compute the current step's output. They adopted the same masking mechanism as Bello et al. [47] to mask visited nodes, ensuring that the obtained solution is feasible.

The algorithm employs a classic policy gradient algorithm in reinforcement learning, namely REINFORCE algorithm [114], to train model. Besides, it improves a baseline by deriving it from the current best policy model during training through deterministic greedy rollout, aiming to reduce gradient variance and accelerate training speed.

Specifically, The REINFORCE algorithm focuses on updating the policy parameters $\boldsymbol{\theta}$ to maximize the expected reward. The key formula utilized in their method is presented as follows [50]:

$$\nabla \mathcal{L}(\boldsymbol{\theta} \mid s) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{\pi}|s)}\left[(L(\boldsymbol{\pi}) - b(s)) \nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{\pi} \mid s)\right], \quad (4)$$

where $\mathcal{L}(\boldsymbol{\theta} \mid s)$ denotes the loss function of the policy; $p_{\boldsymbol{\theta}}(\boldsymbol{\pi} \mid s)$ represents the probability distribution of the policy π given the state s; $L(\boldsymbol{\pi})$ signifies the reward of the trajectory $\boldsymbol{\pi}$, which, in the context of TSP, corresponds to the total length of the tour; and $b(s)$ acts as a baseline function employed to reduce variance, calculated as the cost of a solution from a deterministic greedy rollout of the policy defined by the best model so far.

The Adam [115] optimizer is then utilized to update the parameters as follows:

$$\boldsymbol{\theta} \leftarrow \text{Adam}(\boldsymbol{\theta}, \nabla \mathcal{L}). \quad (5)$$

The algorithm is also tested on TSP20, TSP50, and TSP100 datasets. In comparison with heuristic algorithms and previous DL-based algorithms [46–49], this algorithm demonstrates a noteworthy enhancement in solution quality.

● POMO framework

In 2020, Kwon et al. [51] proposed a policy optimization with multiple optima (POMO) framework, building upon Kool et al. algorithm [50], to tackle the symmetry issue in TSP, as illustrated in Fig. 13.

The algorithm employs a reasoning strategy based on multiple greedy rollouts to simultaneously generate multiple tours $\{\tau^1, \tau^2, \ldots, \tau^N\}$ with varying starting nodes. This simultaneous exploration of TSP's symmetry proves more

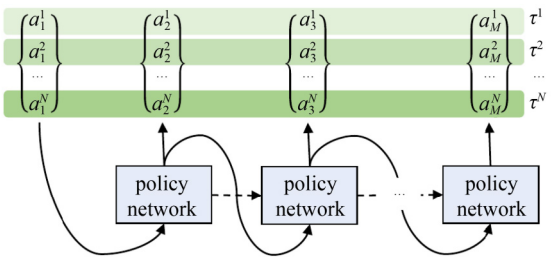

Fig. 13 POMO framework [51]

effective than traditional sampling inference methods. Additionally, the algorithm devises a new low-variance baseline.

POMO framework is evaluated on TSP20, TSP50, and TSP100 datasets too, showcasing its superiority over OR-Tools [113] and previous DL-based algorithms [50,62,63,65].

● Transformer + RL algorithm

In 2021, Bresson and Laurent [52] proposed a framework based on transformer for TSP. The framework employs a standard transformer architecture as the encoder and an attention-based decoder to construct tours in an auto-regressive manner. The model is trained using reinforcement learning.

Several innovations are introduced to enhance model's performance. A position encoding scheme tailored for TSP is incorporated, and a beam search strategy is employed during the decoding phase to refine the quality of the generated tours.

The algorithm is evaluated on TSP50 and TSP100 datasets. Experimental results demonstrate the effectiveness of this Transformer-based method, showing significant improvements in both solution quality and computational efficiency.

● H-TSP algorithm

In 2023, Pan et al. [53] proposed H-TSP, an end-to-end learning framework based on transformer for solving the large-scale TSP. H-TSP, based on hierarchical reinforcement learning, comprises two components: an upper-level policy that selects a small subset of nodes to traverse (up to 200 in experiments), and a lower-level policy that takes these selected nodes as input and outputs a journey connecting them to the existing partial route. Following joint training of the upper and lower-level policies, the method can directly generate solutions for given TSPs without resorting to time-consuming search procedures.

Extensive experiments on randomly generated TSP

instances with varying numbers of nodes demonstrate the effectiveness of H-TSP. Results indicate that H-TSP achieves comparable performance to state-of-the-art (SOTA) search methods, while significantly reducing time consumption by two orders of magnitude. H-TSP achieves problem-solving scalability up to 10000 nodes for TSP.

● Sparsification algorithm

In 2024, Lischka et al. [54] introduced a novel approach emphasizing the importance of sparsification for transformers and GNNs in solving TSP.

Their approach involves a data preprocessing method that enables encoders to focus on the most relevant parts of TSP instances. Specifically, they employ graph sparsification for TSP graph representations when using GNNs and attention masking for TSP instances with transformers, aligning the masks with the adjacency matrices of the sparse TSP graphs. Furthermore, they propose using ensembles with varying levels of sparsification, enabling the models to focus on the most promising segments while still ensuring information flow between all nodes of a TSP instance.

Experimental studies demonstrate that appropriate sparsification and ensembles of different sparsification levels lead to substantial performance increases in GNNs, while ensembles of attention masking improve performance in transformers.

● LEHD model

In 2024, Luo et al. [55] proposed the light encoder and heavy decoder (LEHD) model, based on attention, as illustrated in Fig. 14. The algorithm aims at addressing a crucial issue in neural combinatorial optimization: inadequate model generalization.

The LEHD model, with its robust generalization capability, dynamically captures relationships among all available nodes in problems of varying scales, facilitating generalization across problem sizes. Additionally, the authors propose a data-efficient training scheme and a flexible solution construction mechanism for the LEHD model, enabling training on small-scale instances and generating near-optimal solutions for TSP. The LEHD model also demonstrates good generalization performance in solving real-world TSPs.

The LEHD model achieves problem-solving scalability up to 1000 nodes for TSP.

4.3.3 GNN-based algorithms

TSP instances can be represented as graph-structured data. Therefore, GNNs have also been applied to solve TSP.

● S2V-DQN framework

In 2017, Dai et al. [56] explored the fundamental graph structure of TSP using GNN, integrating reinforcement learning to acquire heuristic algorithms for addressing TSP.

Their algorithm adopts a greedy-based solving framework, termed Structure2Vec deep Q-learning (S2V-DQN), illustrated in Fig. 15. Specifically, they utilize a GNN named Structure2Vec (S2V) [88] to parameterize the evaluation function within the greedy-based framework and apply deep Q-learning (DQN) [116] to train S2V.

At each node selection step, S2V generates embeddings for

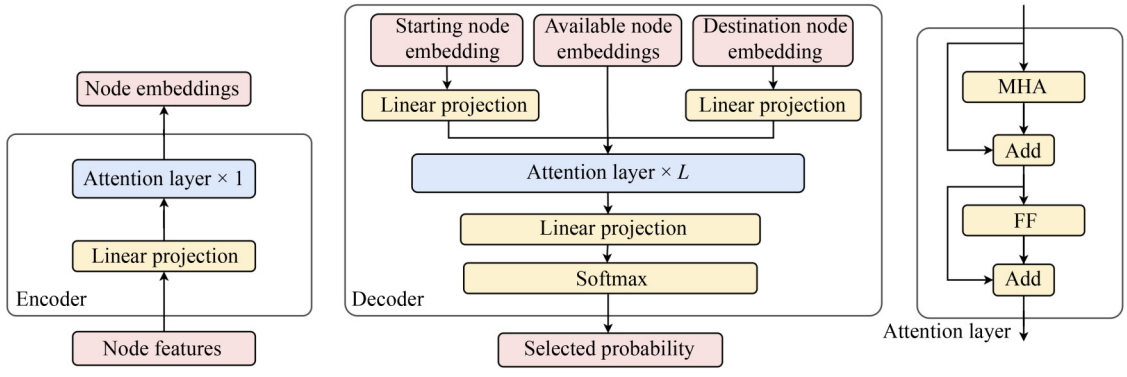

Fig. 14 LEHD model [55]

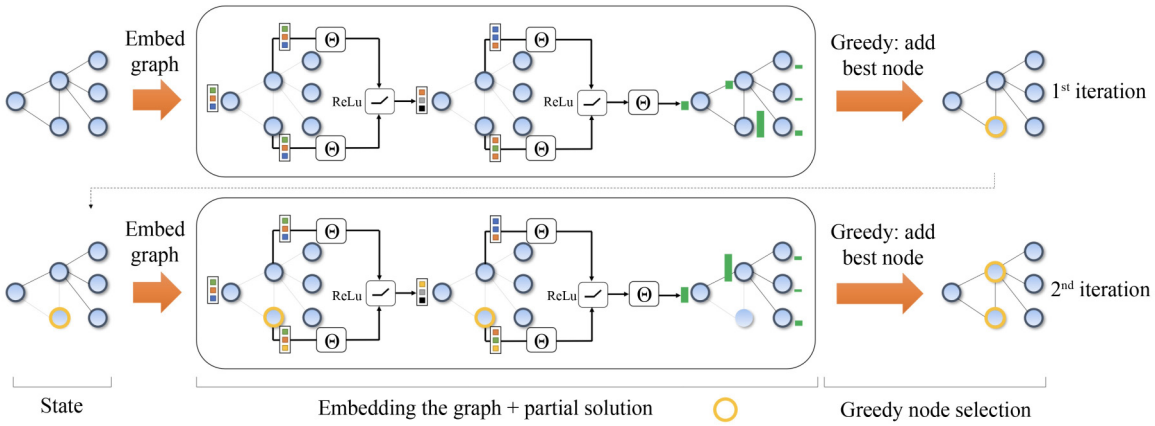

Fig. 15 S2V-DQN framework [56]

each node based on the graph's topology and the neighborhood information of each node, evaluates the Q-values of optional nodes, and then greedily selects a new node to add to the partial solution based on the Q-values, until a complete solution is obtained.

The reason for adopting DQN to train the model is that this approach allows for updating embeddings at each step of the solving process based on partial solutions, without the need to update model parameters only after obtaining a complete solution, as in policy gradient methods.

The algorithm is evaluated on TSP instances ranging from TSP15 to TSP300, with experimental findings indicating that solutions produced by S2V-DQN surpass those of the NCO framework [47] and several heuristic algorithms. Additionally, generalization experiments are conducted, where models trained on TSP50 to TSP100 instances are applied to tackle TSP1000 to TSP1200 instances, resulting in promising approximation ratio to the optimal solution.

● GPN algorithm

In 2019, Ma et al. [57] introduced the graph pointer network (GPN), integrating GAT with the Ptr-Net, aimed at tackling large-scale TSP and TSP with time window (TSPTW) instances.

As depicted in Fig. 16, this algorithm also adopts an encoder-decoder structure. The encoder comprises two components: a node encoder (utilizing LSTM) and a graph encoder (employing GAT), aimed at obtaining node embeddings. The decoder utilizes an attention mechanism similar to compute the probability distribution of node at each step for selection.

This algorithm employs reinforcement learning for model training. Furthermore, the algorithm collaborates with 2-OPT to further bolster solution quality.

Its solution quality slightly surpasses that of Ptr-Net [46] and S2V-DQN [56] on TSP20 and TSP50 datasets, yet falls behind Kool et al. [50]. Additionally, it evaluates models trained on small-scale TSPs to solve large-scale TSPs, demonstrating superior solution quality compared to Ptr-Net [46] and Kool et al. [50], although slightly inferior to S2V-DQN [56].

● GAT + Attention algorithm

In 2020, Drori et al. [58] introduced a framework based on GAT.

This framework adopts an encoder-decoder architecture. The encoder employs GAT to obtain node embeddings, while the decoder utilizes attention mechanism to compute the probability distribution of nodes at each time step for selection.

This algorithm employs reinforcement learning for model training.

The algorithm was evaluated on TSP100 dataset. Results indicate that well-trained models can obtain approximate solutions in linear runtime, outperforming S2V-DQN [56] and GPN [57] in solution quality. Additionally, the algorithm was tested on solving TSP250 using models trained on TSP100 randomly generated dataset and on real-world TSP instances ranging in scale from 51 to 442 sourced from TSPLIB [117]. Experimental findings demonstrate superior generalization ability compared to S2V-DQN [56] and GPN [57].

● GNN + pointing algorithm

In 2021, Ouyang et al. [59] addressed the rotational, reflectional, scaling, and translational invariance of input coordinates.

This algorithm also adopts an encoder-decoder structure, with the encoder utilizing GNN and the decoder employing a pointing mechanism. They first augmented the coordinates through rotation, reflection, scaling, and translation and then inputted the data into the encoder, using relative coordinates during encoding to ensure invariance.

This algorithm is trained on randomly generated datasets of up to TSP50 scale, and subsequently applied to solve randomly generated TSP instances ranging from size 20 to 1000, as well as real-world instances ranging from size 51 to 1002. Experimental results demonstrate its robust generalization capability.

4.3.4 Generation-based algorithms

The diffusion model [118] is a probabilistic generative framework inspired by non-equilibrium statistical physics. It deconstructs data distribution structures through iterative forward diffusion and learns reverse diffusion to restore them, yielding a flexible and tractable generative model for rapid

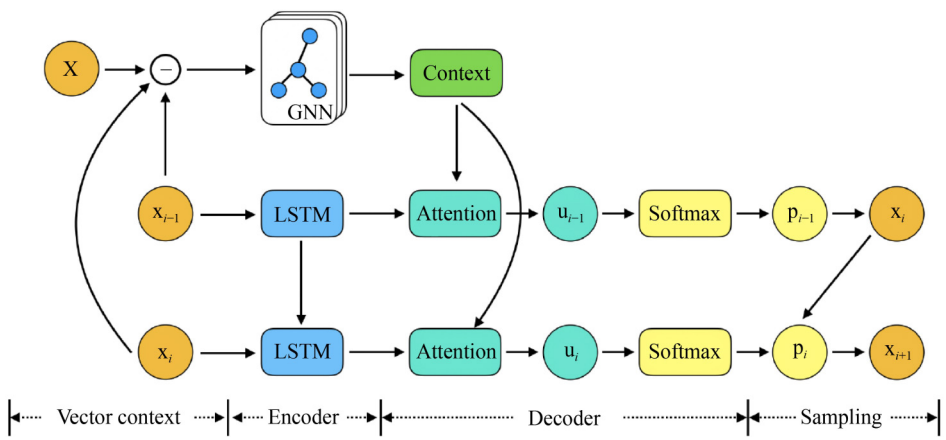

Fig. 16 GPN algorithm [57]

learning, sampling, and probability evaluation in deep generative models. In the field of image processing, the diffusion model has achieved significant success. Therefore, diffusion models have been introduced to the field of combinatorial optimization, specifically for TSP.

● DIFUSCO algorithm

In 2023, Sun and Yang [60] proposed a novel approach to tackle combinatorial optimization problems by proposing a graph-based diffusion solver called DIFUSCO.

DIFUSCO transforms combinatorial optimization problems into discrete vector optimization problems and utilizes graph-based denoising diffusion models to generate high-quality solutions. The study investigates two types of diffusion models with Gaussian and Bernoulli noise and devises effective inference strategies to enhance solution quality.

Experimental results show that DIFUSCO achieves a scale of 10,000 in solving TSP and exhibits superior performance compared to previous neural solvers across benchmark datasets of diverse sizes. DIFUSCO offers a novel approach and solution for efficiently and effectively tackling large-scale TSP.

4.3.5 Adversarial algorithms

In 2023, Liu et al. [61] propose a novel algorithm named adversarial instance augmentation to enhance the generalization of exact solvers (AdaSolver) for combinatorial optimization problems.

AdaSolver aims to address the limited diversity in training distributions, which often leads to severe performance degradation of learning-based solvers on unseen instances, especially those from perturbed environments.

AdaSolver leverages bipartite graph representations for problem instances and obtains various perturbed instances to regularize the solver by augmenting graph structures with a learned augmentation policy. The key technical contribution lies in formulating non-differentiable instance augmentation as a contextual bandit problem and adversarially training the learning-based solver and augmentation policy.

Experiments demonstrate that AdaSolver significantly improves the generalization of exact solvers across various distributions by producing diverse augmented instances.

4.3.6 Summary

We systematically introduce DL-based end-to-end construction algorithms according to diverse model architectures they employ, which include Ptr-Net-based, Transformer-based, GNN-based, generation-based and adversarial algorithms.

As early as 2015, Vinyals et al. [46] proposed Ptr-Net for tackling TSP based on Seq2Seq model and attention mechanism, providing a new direction for subsequent research. However, this algorithm, relying on supervised learning for model training, is limited by the need for large amounts of labeled training data, constraining both solution scalability and quality. In 2016, Bello et al. [47] proposed NCO framework, utilizing reinforcement learning to train Ptr-Net. This algorithm, not requiring labeled data, learns through interaction with the environment. It further improves solution quality through strategies like active search, outperforming supervised learning methods. In 2018, Nazari et al. [48] simplified Ptr-Net by removing LSTM, making the model insensitive to node sequence order and reducing training time.

With the emergence of the Transformer model, Deudon et al. [49] and Kool et al. [50] separately proposed Transformer-based models, although with differences. In 2020, Kwon et al. [51] explored the symmetry issue in TSP using a parallel approach based on the model by Kool et al. [50]. In 2021, Bresson and Laurent [52] introduced a transformer-based framework. In 2023, Pan et al. [53] proposed H-TSP for addressing large-scale TSP instances. In 2024, Lischka et al. [54] presented a novel approach highlighting the significance of sparsification for transformers for TSP. Additionally, in 2024, Luo et al. [55] introduced the LEHD model. Algorithms based on transformer exhibit high solution quality, parallelizability, and good generalization performance.

In 2017, Dai et al. [56] explored TSP using GNN, developing S2V-DQN algorithm combined with reinforcement learning. S2V-DQN constructs a parameterized greedy policy trained through reinforcement learning, exhibiting excellent performance and generalization capability. In 2019, Ma et al. [57] proposed GPN, integrating GNN and Ptr-Net to tackle large-scale TSP and TSPTW. In 2020, Drori et al. [58] proposed a framework based on GAT. In 2021, Ouyang et al. [59] addressed the issue of input invariance, augmenting input through rotation, flipping, scaling, and translation, and utilized GNN to tackle TSP, exhibiting strong generalization capability. In 2024, Lischka et al. [54] presented a novel approach highlighting the significance of sparsification for GNNs for TSP.

In 2023, Sun and Yang [60] introduced DIFUSCO, a graph-based diffusion solver, tailored for resolving large-scale TSP instances.

Also in 2023, Liu et al. [61] proposed AdaSolver, aimed at improving the generalization capabilities of exact solvers for combinatorial optimization problems.

In summary, these algorithms provide diverse approaches to construct TSP solutions, integrating SL or RL, and embracing attention mechanism, Ptr-Net, Transformer, GNN, generative and adversarial models. Each algorithm exhibits distinct strengths. However, these algorithms construct feasible solutions once and for all, leading to rapid solving speeds. Nonetheless, apart from a few exceptions, the majority of algorithms in this category produce solutions of relatively low quality.

4.4 DL-based end-to-end improvement algorithms

Since the advent of Ptr-Net, research on solving TSP using DL has primarily focused on end-to-end construction algorithms. Although these construction algorithms can rapidly obtain approximate solutions, the gap between their solutions and the optimal remains significant. Subsequently, end-to-end improvement algorithms have emerged with considerable potential to narrow this gap, yielding higher-quality solutions. However, compared to construction algorithms, improvement algorithms are more time-consuming.

DL-based end-to-end improvement algorithms typically employ deep reinforcement learning to learn improvement policies, such as k-OPT heuristics.

As introduced in Section 3.2, the k-OPT algorithm is a local search method used for TSP. The core concept involves removing k edges from a feasible tour and attempting to reconnect the remaining segments to find a shorter path. This improvement process is iteratively performed to obtain a high-quality solution. Its advantages are significant: k-OPT can substantially reduce tour length and improve solution quality; it is also easy to implement. However, its disadvantages include the exponential growth in the number of possible reconnection combinations as k increases, leading to higher computational complexity. As mentioned in Section 3.2, 2-OPT has only one reconnection type, while 3-OPT has seven reconnection types. Additionally, k-OPT may get trapped in local optima and cannot guarantee finding the global optimum.

Using DL can not only identify which k edges to remove but also select the most appropriate reconnection type from exponentially many possibilities. Therefore, DL-based end-to-end improvement algorithms typically learn the k-OPT heuristic to enhance solutions.

Specifically, these algorithms [41,62–64] also typically adopt an encoder-decoder structure. The encoder generates embeddings for nodes (or edges) and the current tour, while the decoder selects k inferior edges based on the embeddings, removes these k edges from the current tour, and adds k superior edges to reconnect the remaining segments, thereby obtaining a higher-quality solution. This process iterates multiple times until a high-quality solution is found.

4.4.1 2-OPT-based algorithms

● Learned 2-OPT algorithm by Wu et al.

In 2019, Wu et al. [62] leveraged deep reinforcement learning to learn an improvement policy based on 2-OPT heuristic.

The algorithm devised a model based on self-attention mechanism, comprising a node embedding module and a node pair selection module.

As illustrated in Fig. 17, the node embedding module takes node features and sequence positions as input to generate node embeddings. Subsequently, these embeddings are fed into the node pair selection module, which outputs a probability matrix. Each element in the matrix represents the probability of selecting the corresponding pair of nodes at the respective row and column positions.

During each iterative improvement step, two edges are removed from current solution based on the output node pairs, and two new edges are added to obtain a new solution. This iterative process continues until a high-quality solution is attained.

The algorithm was evaluated on TSP20, TSP50, and TSP100 datasets, demonstrating superior solution quality compared to that of Kool et al. [50] and OR-Tools solver [113]. Additionally, the algorithm tested models trained on small-scale TSPs to solve larger-scale TSPs, showing favorable generalization performance.

However, since the method can only select pairs of nodes, specifically applicable to 2-OPT, it faces challenges in extending to k-OPT ($k \geqslant 3$) strategies.

● Learned 2-OPT algorithm by Costa et al.

In 2020, Costa et al. [63] also employed deep reinforcement learning to learn an improvement heuristic based on 2-OPT.

Their model also follows an encoder-decoder structure, as depicted in Fig. 18.

The encoder comprises a GCN and two LSTMs. The GCN encodes the input graph's topology, while the two LSTMs encode the current solution sequence from both forward and backward directions. Additionally, another set of encoders with identical structures is employed to encode the input graph and the current best solution. The final encoder outputs embeddings for nodes and solution sequences (including the current solution and the current best solution).

The decoder consists of a policy decoder and a value

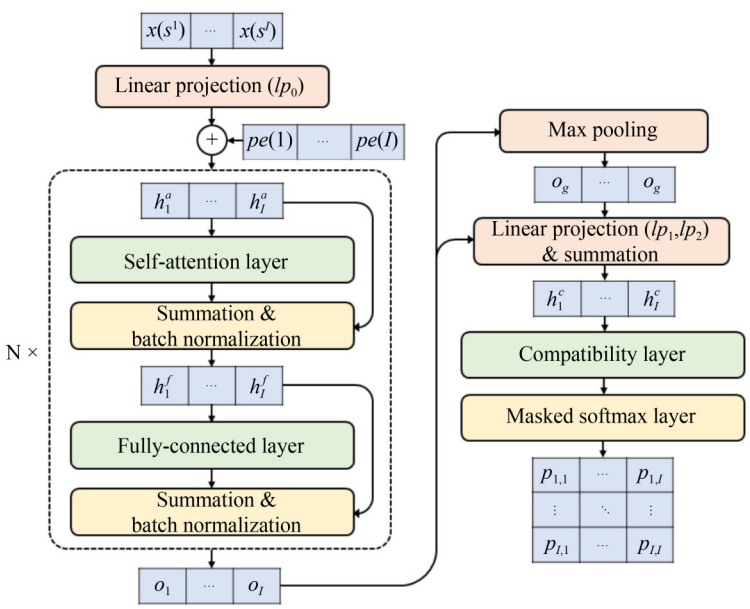

Fig. 17 Learned 2-OPT algorithm by Wu et al. [62]

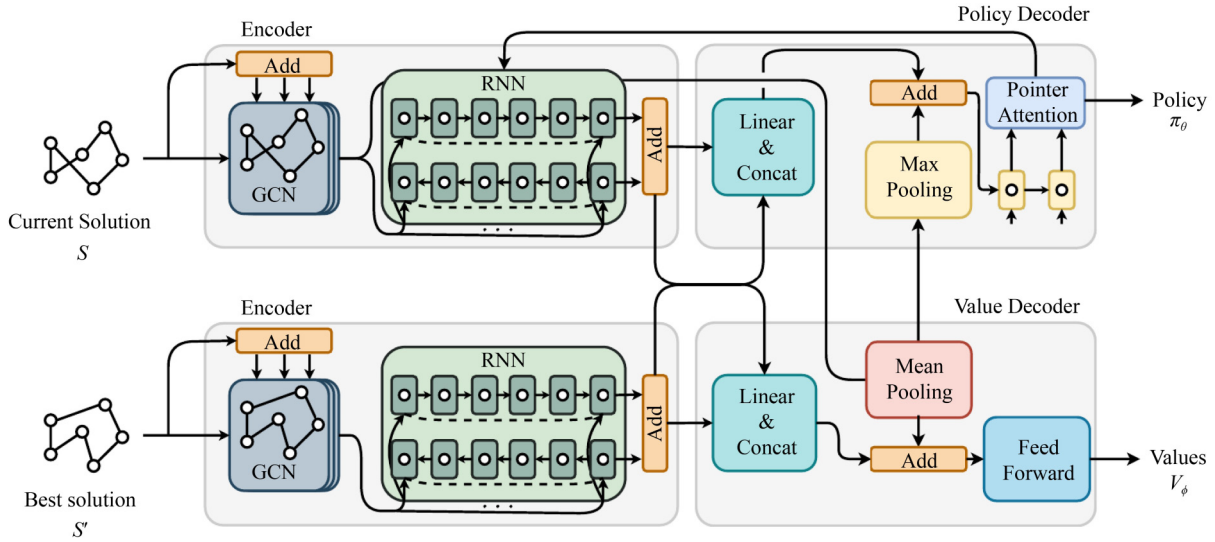

Fig. 18 Learned 2-OPT algorithm by Costa et al. [63]

decoder. The policy decoder adopts a pointer mechanism to sequentially output two actions (i.e., two nodes representing the positions of two edges). Subsequently, based on the output node positions, two edges are removed from current solution, and two new edges are added to obtain a new solution. The value decoder is used to estimate the value of the current state for model training.

The algorithm employs policy gradient algorithm to train model.

The algorithm's performance is evaluated on TSP20, TSP50, and TSP100 datasets. It outperforms construction algorithms such as Ptr-Net [46], Bello et al. [47], Kool et al. [50], and S2V-DQN [56], as well as Wu et al.'s [62] improvement algorithm and Google's OR-Tools solver [113].

It is worth noting that although the algorithm claims to be easily extendable to k-OPT ($k \geqslant 3$), it is not straightforward to implement in practice. While the method can output k nodes in the same manner as outputting two nodes, it does not specify how to reconnect the remaining k segments after removing k edges.

● DACT algorithm

In 2021, Ma et al. [64] also leveraged deep reinforcement learning to learn an improvement policy based on 2-OPT.

As illustrated in Fig. 19, this algorithm also adopts an encoder-decoder structure.

In its encoder, a novel Transformer-based model termed dual-aspect collaborative Transformer (DACT) is proposed to separately learn embeddings for node and position features, avoiding potential noise and incompatible correlations by not merging them together as existing Transformers do.

Similar to Wu et al. [62], its decoder outputs the probability distribution of node pairs represented in matrix form.

This algorithm employs proximal policy optimization for model training and incorporates curriculum learning strategies to enhance sample efficiency.

The algorithm is evaluated on TSP20, TSP50, and TSP100 datasets, showing superiority over existing Transformer-based improvement algorithms at the time, with promising generalization capabilities.

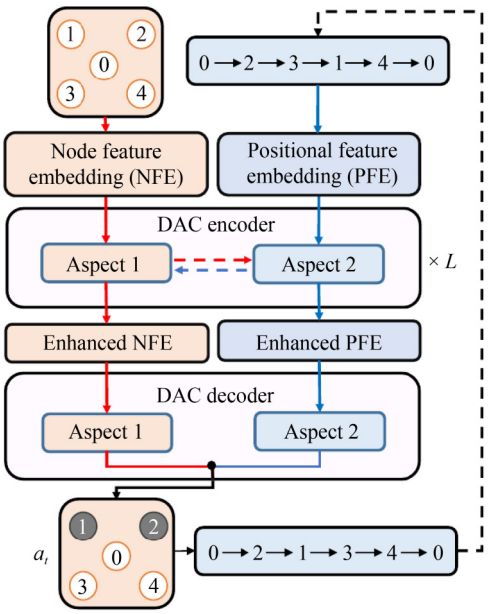

Fig. 19 DACT algorithm [64]

4.4.2 3-OPT-based algorithms

● Neural-3-OPT algorithm

In 2021, Sui et al. [41] employed deep reinforcement learning to learn an improvement heuristic based on 3-OPT, termed Neural-3-OPT.

The main distinction between 3-OPT and 2-OPT lies in the difficulty of reconnecting the remaining segments after removing the corresponding number of edges. While 2-OPT has only one reconnection type, 3-OPT offers up to seven, posing a challenge for the model to select the appropriate one.

To overcome this challenge, Sui et al. combined GCN, LSTM, pointer mechanism, and feature-wise linear modulation network (FiLM-Net) [119] to design a comprehensive model for predicting three edges to be removed and the best matching reconnection type.

As depicted in Fig. 20, this model also follows an encoder-decoder structure. The encoder utilizes sparse GCN and

bidirectional LSTMs to encode TSP graph's topology and solution sequence. The decoder consists of an edge selector and a reconnection type selector. The edge selector adopts a pointer mechanism to select three edges, while the reconnection type selector utilizes FiLM-Net to fuse the outputs of the encoder and the edge selector for selecting the reconnection type. This algorithm employs the learned improvement policy to iteratively enhance an initial solution until reaching the predefined number of iterations.

Neural-3-OPT algorithm also employs policy gradient algorithm for training model.

Neural-3-OPT algorithm is evaluated on TSP20, TSP50, and TSP100 datasets, outperforming previous DL-based end-to-end construction and improvement algorithms. Additionally, the algorithm demonstrates excellent generalization by testing models trained on small-scale TSPs to solve larger-scale TSPs. Furthermore, empirical evidence shows that among the seven reconnection ways, random, fixed, and greedy strategies are not the most effective; instead, the predicted reconnection type by the model yields better solutions.

4.4.3 Summary

DL-based end-to-end improvement algorithms learn heuristics based on k-OPT.

Algorithms proposed by Wu et al. [62], Costa et al. [63], and Ma et al. [64] all incorporate the 2-OPT heuristic, although with different model architectures. While the models by Wu et al. [62] and Ma et al. [64] output probabilities for pairs of nodes, making them challenging to extend to larger k-OPT, Costa et al.'s [63] algorithm can sequentially output two nodes, offering potential for extension to larger k-OPT. However, they do not address the challenge of selecting reconnection types when $k \geqslant 3$.

Sui et al.'s [41] Neural-3-OPT algorithm learns policy based on 3-OPT heuristic. Besides, when $k \geqslant 3$, their proposed model framework can sequentially predict k edges and identify the most suitable reconnection type among multiple options. Hence, it can be extended to larger k-OPT heuristics, laying the groundwork for designing more complex model frameworks and blending multiple k-OPT heuristics in the future. Experimental results demonstrate that, under the same number of iterations, learned heuristics based on 3-OPT outperform those based on 2-OPT in finding higher-quality solutions.

4.5 Direct hybrid algorithms

Direct hybrid algorithms have emerged as an effective way for tackling TSP. While DL-based end-to-end construction algorithms can rapidly generate solutions, they often struggle to achieve higher-quality solutions. On the other hand, DL-based end-to-end improvement algorithms, while enhancing solution quality, typically incur longer solving times. DL possess learning capabilities but lack search powers, whereas search algorithms such as local search [42], LKH [43] have demonstrated robust search capabilities but lack learning capabilities. Combining DL with search algorithms has the potential to achieve superior solving performance.

In recent years, there has been a series of hybrid algorithms directly combining DL and heuristic algorithms to solve TSP, gradually demonstrating stronger solving and generalization capabilities compared to using either alone. These algorithms include combinations of DL with beam search (BS) [65], guided local search (GLS) [66,67], monte carlo tree search (MCTS) [68], Ising heuristic (Ising) [69], and LKH (LKH-3) [70–72].

4.5.1 DL combined with beam search

● GCN + BS algorithm

In 2019, Joshi et al. [65] proposed an algorithm that combines DL with beam search for TSP.

As illustrated in Fig. 21, this algorithm employs residual gated graph convolutional neural network [92] to encode TSP graph structure, producing a probability matrix (akin to a heat map of the adjacency matrix), where each element represents the probability of corresponding edge appearing in the solution. Subsequently, an effective solution is obtained using beam search based on this probability matrix.

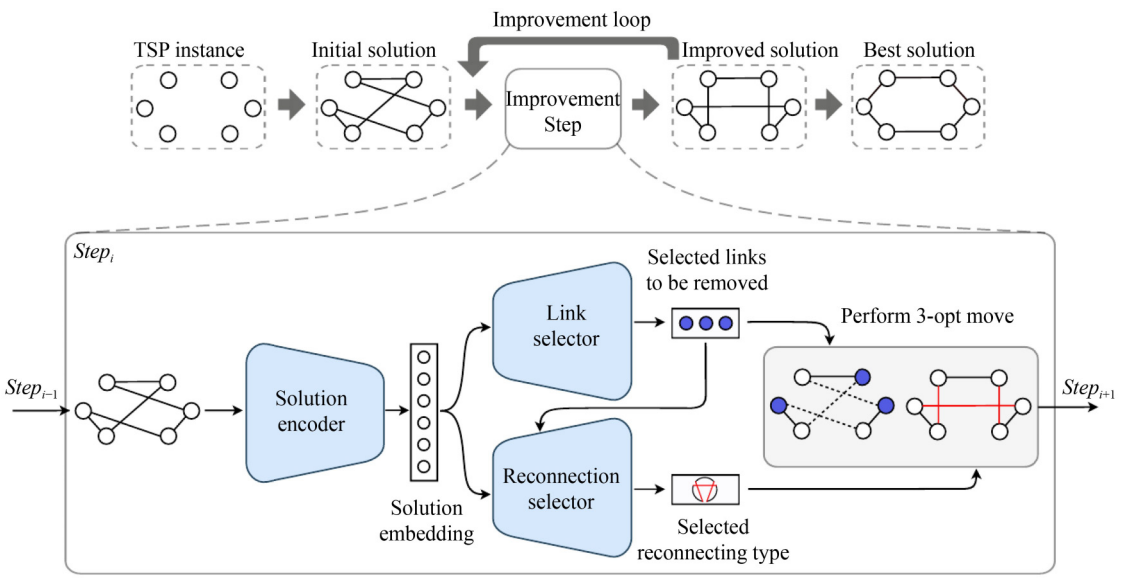

Fig. 20 Neural-3-OPT algorithm [41]

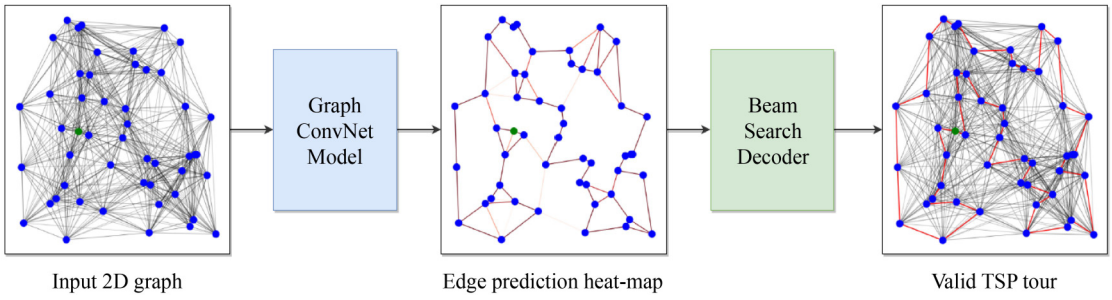

Fig. 21 GCN+BS algorithm [65]

All components of this algorithm are highly parallelizable, giving it a performance advantage. However, compared to autoregressive models, this non-autoregressive approach of generating solutions at once exhibits poorer generalization.

The model is trained using supervised learning.

It was tested on TSP20, TSP50, and TSP100, demonstrating superior solution quality compared to existing DL-based algorithms at the time.

It's notable that Joshi et al. [65] utilize a GCN to generate an edge adjacency matrix, followed by beam search to convert this matrix into a valid tour, playing a pivotal role in the solution process. This process is not in end-to-end manner. In contrast, Ptr-Net [46] and Bresson & Laurent [52] construct solutions in an end-to-end manner. While they also employ beam search, its primary role is to filter out invalid solutions or refine solutions, rather than directly constructing the solution with beam search.

4.5.2 DL combined with guided local search

● GAT + GLS algorithm

In 2021, Hudson et al. [66] proposed an algorithm that combines GAT with GLS for TSP.

As depicted in Fig. 22, this algorithm transforms a given TSP graph into a line graph (where nodes represent edges of the original graph) and employs a GAT on the line graph to compute regret value for each edge. These predicted regret values guide the search process of GLS.

The model is also trained using supervised learning.

The algorithm was evaluated on TSP20, TSP50, and TSP100 datasets, demonstrating superior solution quality compared to existing DL-based algorithms at the time.

However, the transformation of a TSP graph into a line graph increases the graph size from n to $n(n-1)/2$, adding computational overhead and complexity, which is not favorable for handling large-scale TSPs.

● NeuralGLS algorithm

In 2024, Sui et al. [67] introduced NeuralGLS, an algorithm that integrates GCN with GLS.

As shown in Fig. 23, NeuralGLS includes an adaptive GCN that can consider neighborhoods of different sizes during convolution, better adapting to various scales of TSP instances. This GCN predicts a regret value for each edge in the given TSP instance, which is used to guide the subsequent GLS process for iterative improvement of initial solutions until the stopping condition is met.

Experimental results indicate that NeuralGLS not only generates high-quality solutions within reasonable computation time but also exhibits good generalization performance on real-world and larger-scale TSP instances. Compared to Hudson et al.'s algorithm, which also utilizes GLS, NeuralGLS exhibits superior solution performance within an equivalent computation time.

4.5.3 DL combined with monte carlo tree search

● Att-GCRN + MCTS algorithm

In 2021, Fu et al. [68] proposed an algorithm that combines DL with MCTS for TSP, as illustrated in Fig. 24.

The algorithm employs supervised learning to train an attention mechanism-equipped graph convolutional residual network (Att-GCRN) on small-scale problems to generate a heat map (representing the probability of edges being selected in the solution). Subsequently, this network, along with techniques such as graph sampling, graph transformation, and heat map merging, constructs a heat map for large-scale TSP problems. The information stored in the heat map guides MCTS to find solutions.

The algorithm's generalization ability was evaluated by training models on TSP20 and applying them to solve TSP20/50/100 instances, as well as training models on TSP50 and using them to solve TSP200/500/1000/10000 instances. Experimental results demonstrate that this algorithm significantly outperforms other DL-based algorithms at the time, showing remarkable generalization capabilities.

4.5.4 DL combined with Ising algorithm

● Neuro-Ising algorithm

In 2022, Sanyal and Roy [69] proposed a framework called

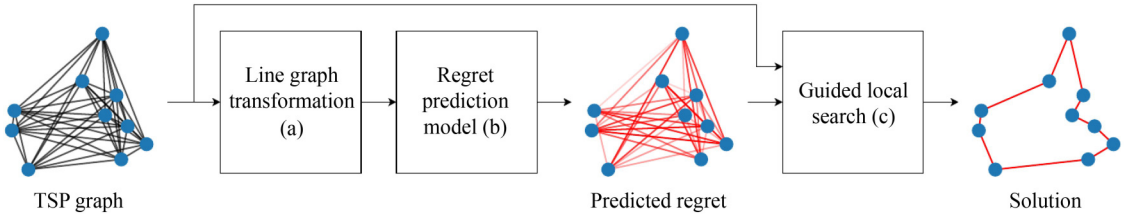

Fig. 22 GAT+GLS algorithm [66]

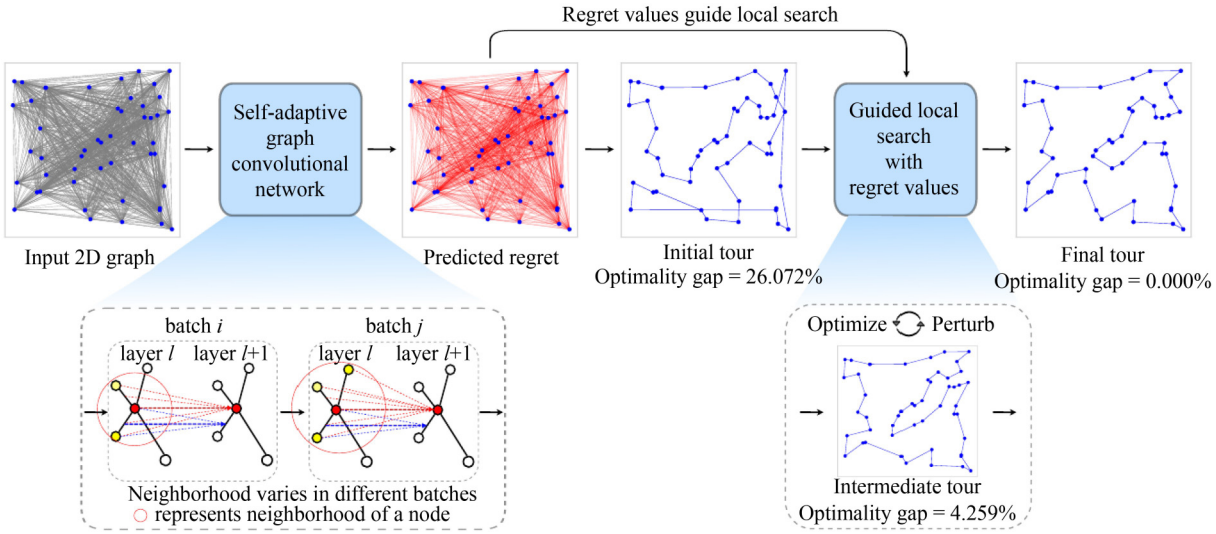

Fig. 23 NeuralGLS algorithm [67]

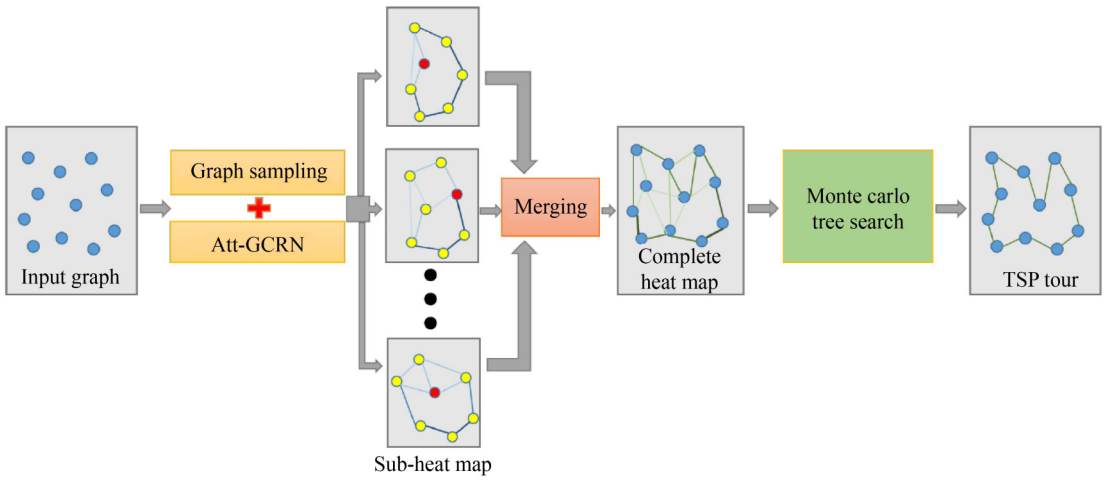

Fig. 24 Att-GCRN+MCTS algorithm [68]

Neuro-Ising, which combines GNN with Ising algorithm for TSP.

Neuro-Ising initially partitions the original problem into smaller sub-problems using clustering. This decomposition divides the problem-solving into two levels: at the lower level, each sub-problem is solved using the Ising algorithm, while at the higher level, a GNN coordinates and combines the local solutions from the lower level to generate a global solution.

Experiments demonstrate that Neuro-Ising accelerates the solution process for TSP compared to Tabu search algorithm [120]. Additionally, when compared with two other solvers based on clustering algorithms, Neuro-Ising exhibits improvements in both solution speed and quality.

4.5.5 DL combined with LKH algorithm

● VSR-LKH algorithm

In 2021, Zheng et al. [70] proposed an algorithm, named variable strategy reinforced LKH (VSR-LKH), which integrates DL with the powerful heuristic algorithm LKH [43].

VSR-LKH employs Q-learning, Sarsa, or Monte Carlo reinforcement learning techniques to enhance LKH's k-OPT process, replacing its inflexible traversal operations with reinforcement learning for selecting appropriate edges to achieve k-OPT.

Moreover, VSR-LKH introduces a variable strategy mechanism to combine the advantages of the three reinforcement learning methods. When VSR-LKH determines that the current reinforcement learning method may no longer optimize the path effectively, the variable strategy mechanism switches to another method.

VSR-LKH was tested on 111 real-world TSP instances, demonstrating superior solution quality compared to the LKH algorithm. However, VSR-LKH conducts reinforcement learning during the search process for each instance, rather than learning general patterns across a class of instances.

In 2023, Zheng et al. [72] extended the VSR-LKH algorithm to develop VSR-LKH-3, which can be applied to solve various variants of TSP.

● NeuroLKH algorithm

In 2021, Xin et al. [71] proposed NeuroLKH, an algorithm that combines DL with LKH, as depicted in Fig. 25. Unlike VSR-LKH, which solves individual instances, NeuroLKH learns general solution patterns across a class of instances.

Specifically, NeuroLKH employs supervised learning to train a sparse graph network on randomly generated TSP datasets to learn edge scoring functions and utilizes unsupervised learning to learn node penalties, both crucial for enhancing LKH's performance. Based on the output of the sparse graph network, NeuroLKH constructs a candidate set of edges and transforms edge distances to guide LKH's search process.

NeuroLKH was tested on both randomly generated and real-world TSP instances, demonstrating significant improvements over LKH and exhibiting excellent generalization to larger-scale and differently distributed TSP instances.

4.5.6 Summary

Direct hybrid algorithms have emerged as a promising approach for solving TSP, demonstrating superior solving and generalization capabilities compared to using either approach alone.

These direct hybrid algorithms can typically find optimal solutions for small-scale TSPs and exhibit robust performance for large-scale and differently distributed TSPs.

Compared to DL-based end-to-end construction and improvement algorithms based solely on DL, direct hybrid algorithms are limited by the need to pre-design search frameworks. Direct hybrid algorithms demonstrate efficacy in addressing previously investigated problems equipped with established search frameworks. However, the challenge arises when it comes to devising search frameworks for unexplored problems lacking established search algorithms.

4.6 LLM-based hybrid algorithms

Recently, LLM-based algorithm design has yielded a series of notable achievements [103–107]. Specifically, methods combining LLMs with evolutionary algorithms to evolve heuristics have emerged. For instance, FunSearch [16] utilizes an evolutionary framework with LLMs to automatically search functions.

4.6.1 LLM-based heuristics evolutionary algorithms

Evolutionary algorithms are a class of optimization algorithms inspired by the process of natural selection and evolution [121]. They work by iteratively searching through a population of candidate solutions, applying principles such as mutation, recombination, and selection to generate new candidate solutions over successive generations.

LLM-based heuristics evolutionary algorithms merges LLMs with evolutionary algorithms, capitalizing on the generation capability and semantic understanding of LLMs to generate and evolve heuristics, thereby enhancing problem-solving efficiency and effectiveness. Typical algorithms for TSP include those proposed by Liu et al. [73], Ye et al. [74], and Liu et al. [75].

● AEL framework

Liu et al. [73] proposed the algorithm evolutionary using large language model (AEL) framework, as shown in Fig. 26.

AEL involves the evolution of a population of individuals and includes fundamental components such as initialization, selection, crossover, mutation, and population management.

In contrast to conventional evolutionary computing algorithms, where individuals correspond to feasible solutions, each individual in the AEL framework represents an algorithm explicitly designed for TSP. AEL evolves algorithms with the capability to generate innovative and competitive search strategies for a given problem, rather than aiming to improve solutions for specific instances. The process of creating and refining algorithms in AEL is automated, thereby eliminating the necessity of training new models or relying on baseline algorithms.

AEL is employed to devise a constructive algorithm for TSP.

Experimental results show that AEL outperforms simple hand-crafted and LLM-generated heuristics. Additionally, AEL exhibits good generalization capabilities across various problem sizes.

● ReEvo framework

Ye et al. [74] proposed reflective evolution (ReEvo), a generic search framework designed to emulate the reflective strategies employed by human experts while capitalizing on scalable LLM inference, extensive domain knowledge, and evolutionary search capabilities.

ReEvo integrates humanoid reflections to enhance LLM reasoning, employing evolutionary computation for exploring heuristic spaces. It incorporates short-term reflections to address disparities in heuristic performance and long-term reflections across iterations, thus simulating human expertise. This dual-level reflection enhances the effectiveness of heuristic search.

Figure 27 illustrates the schematic of ReEvo framework. Within the evolutionary framework, LLMs serve two distinct roles: a generator LLM generates individuals, while a reflector LLM guides the generation process through reflections. ReEvo employs a unique individual encoding, representing

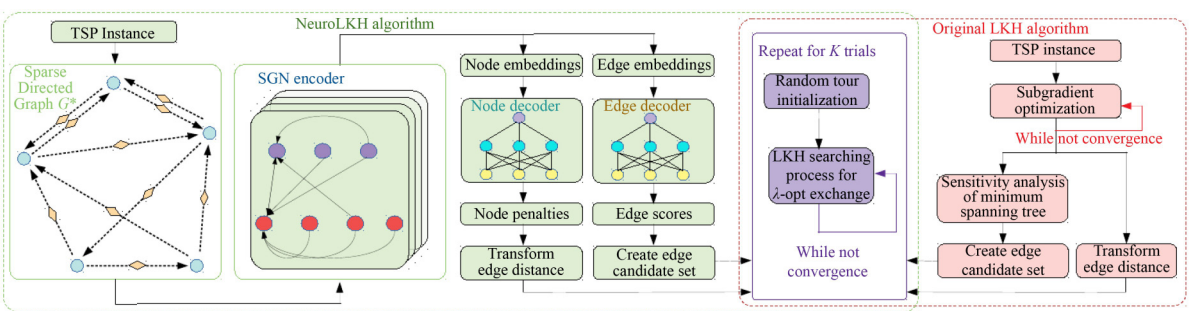

Fig. 25 NeuroLKH algorithm [71]

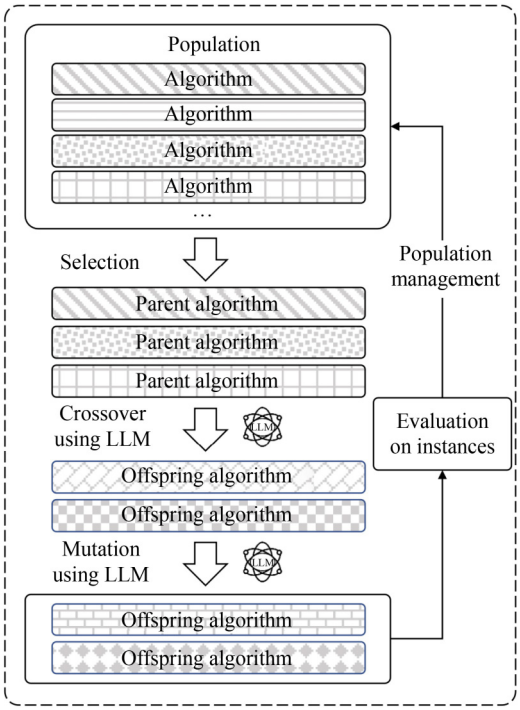

Fig. 26 AEL framework [73]

each individual as a heuristic code snippet. Its evolution begins with population initialization, followed by selection, short-term reflection, crossover, long-term reflection, and elitist mutation iterations. Evaluations are interleaved after both crossover and mutation steps.

ReEvo evolves heuristics for the perturbation of GLS. It seeks the heuristic that yields best performance in GLS for TSP.

Experimental results demonstrate that ReEvo significantly enhances knowledge-guided local search [122] and surpasses state-of-the-art baselines. Additionally, it exhibits robust generalization capabilities.

● EoH algorithm

Liu et al. [75] proposed evolution of heuristic (EoH) paradigm, which combines LLMs and evolutionary computation to facilitate automatic heuristic design.

EoH utilizes LLMs to generate linguistic descriptions, termed thoughts, which are then translated into executable code representations. By co-evolving thoughts and codes within an evolutionary framework, EoH achieves superior performance while mitigating computational expenses.

EoH is employed to design guided heuristics used in the perturbation of GLS for solving TSP. Upon designing each guided heuristic during the evolution, EoH integrates it with fixed local search operators to create a new GLS.

EoH is evaluated on TSP20/50/100 datasets and real-world TSP instances, outperforming DL-based end-to-end construction and improvement algorithms. This algorithm outperforms DL-based end-to-end algorithms and various hybrid algorithms, but performs worse than the hubrid algorithm proposed by Xin et al. [71].

4.6.2 Summary

Recent advances in algorithm evolution leveraging LLMs have opened up new avenues in heuristic design for TSP. These algorithms integrate LLMs with evolutionary techniques to autonomously generate and refine effective heuristics.

The AEL framework proposed by Liu et al. [73] utilizes LLMs to automatically generate constructive algorithms for TSP. The ReEvo framework proposed by Ye et al. [74] emulates human reflective strategies to enhance LLM reasoning, evolving heuristics optimized for GLS perturbation. Similarly, the EoH paradigm proposed by Liu et al. [75] combines LLMs with evolutionary computation to automatically design guided heuristics for GLS perturbation.

In summary, these LLM-based algorithm evolution approaches offer promising avenues for automating heuristic design and evolution without the need for model training, as required by previous DL-based algorithms.

4.7 Trends of research for TSP

The four categories of DL-based algorithms (DL-based end-to-end construction algorithms, DL-based end-to-end improvement algorithms, Direct hybrid algorithms, and LLM-based hybrid algorithms) for TSP are essentially driven by

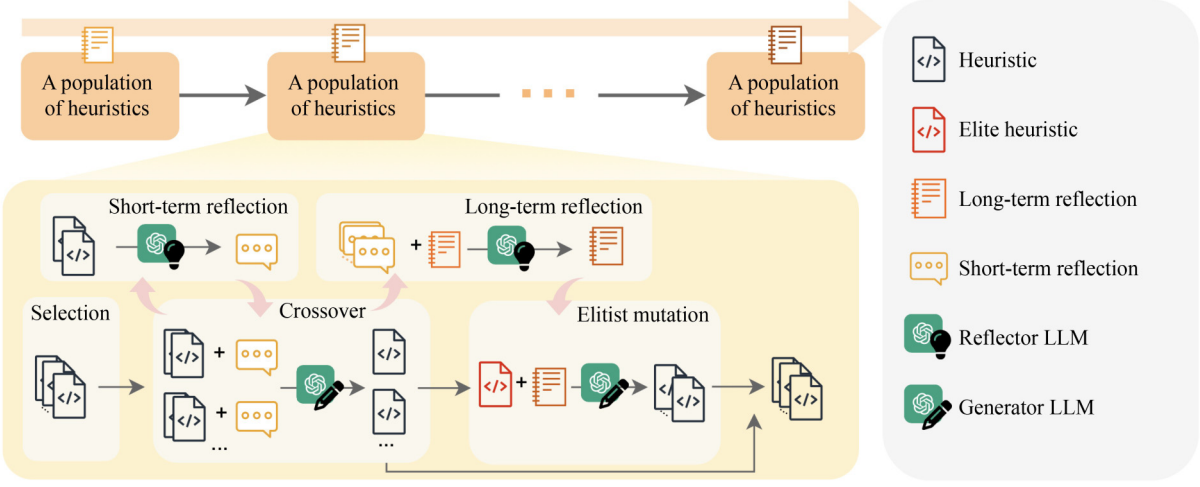

Fig. 27 ReEvo framework [74]

technological advancements. Specifically, with the emergence of Ptr-Net, a wave of DL-based end-to-end construction algorithms was initiated. However, this category of algorithms construct a feasible solution once and for all, resulting in faster but usually lower-quality solutions. Consequently, researchers began exploring DL-based iterative improvements, leading to the development of end-to-end improvement algorithms, which enhance solution quality but sacrifice speed due to the need for repeated complex neural network computations in the iteratively improve process. To address this, researchers integrated DL with traditional search algorithms, leveraging DL to support decision-making and significantly improving both computational quality and efficiency. With the recent rapid advancement of LLMs technology, researchers have begun exploring the integration of LLMs with evolutionary algorithms to generate new heuristics and guide the evolution process.

It's worth noting that the chronological sequence of these four algorithms isn't strictly defined but rather reflects a prominent research trajectory. Despite this overarching trend, occasional emergence of some algorithms within these four categories still occurs.

Currently, hybrid algorithms, particularly those based on LLMs, are gaining increasing attention.

4.8 Algorithms for large-scale TSP

It's notable that some of the algorithms [53,55,57,59,60,68,69,71–73] introduced above are capable of addressing large-scale TSP instances.

These algorithms demonstrate robust capabilities in addressing large-scale TSPs. Regarding problem scale, Pan et al. [53], Sun and Yang [60], and Fu et al. [68] achieved favorable outcomes on randomly generated TSP10000 instances, while Xin et al. [71] achieved excellent results on randomly generated TSP5000 instances. In contrast, Luo et al. [55], Ma et al. [57], Ouyang et al. [59], and Liu et al. [73] attained satisfactory outcomes on randomly generated TSP1000 instances. Furthermore, Sun and Yang's algorithm [60] exhibited inferior performance compared to Fu et al.'s [68] and Pan et al.'s [53] at TSP10000 instances.

As indicated in their respective studies, Sanyal & Roy [69] and Zheng et al. [72] demonstrate strong ability in solving real-world TSP instances sourced from TSPLIB [117]. Sanyal and Roy [69] effectively handle TSPs with up to 4461 nodes, whereas Zheng et al. [72] can tackle instances with up to 85900 nodes.

It's worth noting that these results may be affected by diverse problem instances and might not fully represent algorithm performance. Furthermore, taking into account algorithm attributes and suitability, specific approaches, such as Sun and Yang's algorithm [60], are designed for large-scale TSPs, whereas others, like Zheng et al. [72], exhibit strong applicability in real-world scenarios. These observations underscore the potential of DL-based methods in tackling large-scale TSP instances. However, further research and refinement are needed to overcome current limitations and improve the accuracy and efficiency of problem-solving methodologies.

5 Experimental results and discussion

We carefully selected representative algorithms for evaluation within each category of DL-based algorithms. As a result, among the four categories of DL-based algorithms, we selected a total of six representative algorithms for experimentation.

Specifically, the chosen algorithms encompass Kool et al.'s algorithm [50], representing DL-based end-to-end construction algorithms; Costa et al.'s algorithm [63], representing DL-based end-to-end improvement algorithms; Joshi et al.'s algorithm [65], representing early-stage direct hybrid algorithms; Hudson et al.'s algorithm [66] and Xin et al.'s algorithm [71], representing later-stage direct hybrid algorithms; and Ye et al.'s algorithm [74], representing LLM-based hybrid algorithms.

Researchers typically use both randomly generated and real-world datasets to test algorithms for TSP. Random datasets are usually generated with node coordinates in the unit square $[0,1]^2$, typically of sizes TSP20/50/100, occasionally extended to TSP200 to assess generalization. Real-world TSP datasets come from TSPLIB [117], a library containing 112 symmetric TSP instances with sizes ranging from 14 to 85900, collected from diverse origins and of various types. We summarize the two types of datasets in Table 1.

Therefore, we evaluated the selected representative algorithms using both randomly generated datasets and real-world TSP instances. The randomly generated datasets consist of TSP20, TSP50, TSP100, and TSP200, each containing 10,000 instances with 20, 50,100, and 200 nodes, respectively. Node coordinates are randomly sampled from the unit square $[0,1]^2$. The real-world dataset comprises 29 TSP instances, sourced from TSPLIB [117], offering a diverse range of scales from 51 to 200 and varied distributions, providing a robust dataset for comprehensive analysis.

Additionally, we evaluated their generalization performance by training models on small-scale TSPs and applying them to solve large-scale TSP instances they had not previously encountered.

We quantified solution quality using the optimality gap, defined as the percentage difference between the lengths of solutions obtained by each algorithm and the lengths of exact solutions obtained by the Concorde solver [36].

Additionally, we documented the computation time for each algorithm, reflecting the duration necessary to process individual instance. The values presented in Table 2 denote the average values calculated across the entirety of the test

Table 1 Datasets commonly used to evaluate algorithms performance for TSP

Type	Size	Discription
Randomly generated	20/50/100/200	randomly generated with node coordinates in the unit square $[0,1]^2$
TSPLIB	14 – 85900	from various sources and of various types in the real world

dataset for each TSP scale.

We conducted experiments on a computer with an Intel Xeon Gold 6230 CPU and an NVIDIA GeForce RTX 3090 GPU.

5.1 Evaluation on randomly generated TSPs

Table 2 presents our evaluation of various algorithms' performance on randomly generated TSP instances with different scales (TSP20/50/100).

These algorithms encompass the exact algorithm Concorde [36], four heuristic algorithms, specifically Nearest Neighbor [37], Farthest Insertion [37], Local Search [35], and LKH-3 [45], alongside the six selected DL-based algorithms: Kool et al. [50], Costa et al. [63], Joshi et al. [65], Hudson et al. [66], Xin et al. [71], and Ye et al. [74].

We present the results of the algorithm developed by Kool et al. [50], utilizing the sampling strategy in its decoder. Additionally, we evaluate the algorithm proposed by Costa et al. [63] with 2000 improvement iterations. For both the LKH-3 algorithm [45] and the algorithm introduced by Xin et al. [71], we conduct experiments with different trial settings: 1, 10, and 100, respectively.

Firstly, the exact algorithm Concorde consistently yields optimal solutions across all three TSP datasets. However, as the problem scale increases, there is a noticeable increase in the computational time required by Concorde.

Conversely, heuristic algorithms such as Nearest Neighbor [37] exhibit poorer accuracy but significantly shorter computation times. Farthest Insertion [37] and Local Search [35] algorithms achieve better accuracy than Nearest Neighbor but are more time-consuming. Among the heuristic algorithms, LKH-3 [45] stands out for its ability to find high-quality solutions in short time.

Among the DL-based algorithms, the algorithm proposed by Kool et al. [50], representing a DL-based end-to-end construction algorithm, exhibits higher accuracy than most heuristic algorithms and has relatively shorter computation time. However, compared to LKH-3, its solution quality is lower, and computation time is somewhat slower.

The algorithm proposed by Costa et al. [63], representing a DL-based end-to-end improvement algorithm, achieves significantly better accuracy than construction algorithms but entails longer computation time.

The algorithms proposed by Joshi et al. [65], Hudson et al. [66], and Xin et al. [71], representing direct hybrid algorithms, exhibit higher solution quality compared to heuristic algorithms, albeit with varying solution times. Particularly noteworthy is Xin et al.'s algorithm, which not only achieves significantly higher solution quality compared to heuristic methods and other DL-based algorithms but also demonstrates notably shorter computation time.

The algorithm proposed by Ye et al. [74], representing LLM-based hybrid algorithms, outperforms most heuristic algorithms and DL-based algorithms, with solution quality slightly inferior to Xin et al's hybrid algorithm, while maintaining relatively short computation times.

Figures 28 and 29 illustrate the solution quality and computation time of each algorithms for TSP50 and TSP100 datasets, respectively. Due to significant variations in the numerical values, we transformed the axes to logarithmic scales to enhance readability. Blue and red dots on the graphs represent heuristic algorithms and DL-based algorithms, respectively.

As depicted in both Figs. 28 and 29, hybrid algorithms, particularly Xin et al.'s NeuroLKH [71] algorithm, exhibit superior solution quality, when considering both solution quality and computation time.

Considering both solution quality and computational time, the choice of algorithm may vary depending on the specific requirements of TSPs. For TSPs with strict time constraints, the Nearest Neighbor algorithm may be a suitable choice, while Concorde or LKH-3 may be more appropriate for high-

Table 2 Performance of five traditional algorithms and six classical DL-based algorithms in solving randomly generated TSPs. Gap denotes the optimality gap, measured in percent (%), and Time represents the computation time, measured in seconds (s). C, I, D, and L respectively denotes the four categories of algorithms: End-to-end construction algorithms, End-to-end improvement algorithms, Direct hybrid algorithms, and LLM-based hybrid algorithms

	Algorithm	Category	TSP20		TSP50		TSP100	
			Gap/%	Time/s	Gap/%	Time/s	Gap/%	Time/s
Heuristic	Concorde [36]		0.000	0.010	0.000	0.051	0.000	0.224
	Nearest Neighbor [37]		17.448	0.000	23.230	0.001	25.104	0.006
	Farthest Insertion [37]		2.242	0.002	7.263	0.022	12.456	0.174
	Local Search [35]		1.824	0.007	3.358	0.086	4.169	0.574
	LKH-3 [45] (1 trail)		0.000	0.001	0.009	0.017	0.022	0.050
	LKH-3 [45] (10 trails)		0.000	0.001	0.004	0.021	0.011	0.065
	LKH-3 [45] (100 trails)		0.000	0.002	0.001	0.062	0.002	0.212
DL-based algorithm	Kool et al. [50]	C	0.068	0.026	0.494	0.065	2.359	0.167
	Costa et al.* [63]	I	0.001	0.464	0.134	0.637	0.758	0.931
	Joshi et al. [65]	D	0.035	0.974	0.227	2.261	1.516	4.327
	Hudson et al. [66]	D	0.000	10.008	0.002	10.048	0.454	10.175
	Xin et al. [71] (1 trail)	D	–	–	0.001	0.008	0.009	0.018
	Xin et al. [71] (10 trails)	D	–	–	0.000	0.010	0.003	0.027
	Xin et al. [71] (100 trails)	D	–	–	**0.000**	**0.037**	**0.001**	**0.118**
	Ye et al. [74]	L	0.000	0.007	0.000	0.089	0.004	0.393

* The algorithm is capable of computing a batch of instances simultaneously. To ensure a fair comparison, we present its computation time as the total time divided by batch size, rather than the actual average time per instance. It should be noted that in practice, the algorithm requires significantly more time to compute each instance than what is displayed in the table.

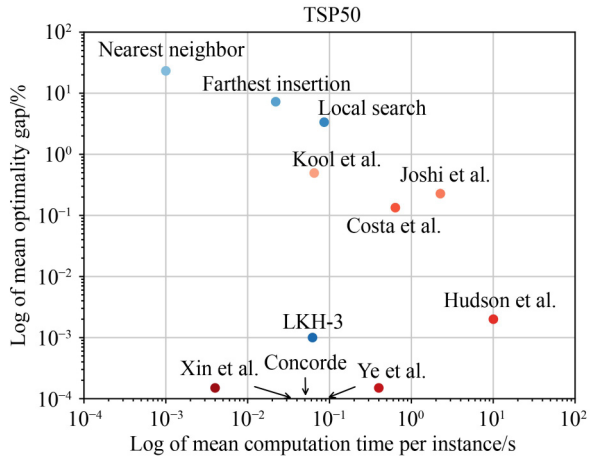

Fig. 28 Comprehensive perspective of solution quality and computation time for various algorithms on TSP50

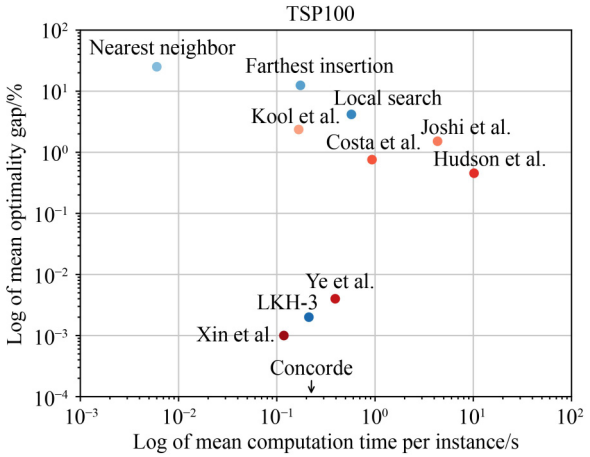

Fig. 29 Comprehensive perspective of solution quality and computation time for various algorithms on TSP100

precision requirements. DL-based algorithms, especially hybrid methods, offer a balanced option between solution quality and computation time. This is because hybrid methods integrate the learning capability of DL with the search capability of heuristic algorithms, enabling them to find better solutions in shorter time frames.

5.2 Evaluation on real-world TSPs

Table 3 displays performance of the six representative DL-based algorithms on solving real-world TSP instances.

In terms of solution quality, the average optimality gap for Kool et al. [50], Costa et al. [63], Joshi et al. [65], Hudson et al. [66], Xin et al. [71] (100 trials), and Ye et al. [75] is 22.659%, 3.976%, 29.775%, 1.318%, 0.013%, and 2.992% respectively. It is evident that hybrid algorithms outperform DL-based end-to-end construction and improvement algorithms in solving real-world TSPs.

Table 3 Performance of six classical DL-based algorithms in solving real-world TSPs

TSP	Kool et al. [50]		Costa et al. [63]		Joshi et al. [65]		Hudson et al. [66]		Xin et al. [71]*		Ye et al. [74]	
	Gap/%	Time/s	Gap/%	Time/s	Gap/%	Time/s	Gap/%	Time/s	Gap/%	Time/s	Gap/%	Time/s
eil51	1.692	0.069	0.394	523.907	8.341	2.414	0.000	10.044	0.000	0.114	2.346	0.001
berlin52	17.953	0.073	1.743	531.19	41.694	2.377	0.000	10.053	0.000	0.070	3.862	0.001
st70	1.930	0.117	1.004	845.866	33.836	3.108	0.000	10.040	0.000	0.083	2.316	0.002
eil76	2.525	0.128	0.693	656.161	5.682	3.432	0.000	10.097	0.000	0.071	1.846	0.001
pr76	2.810	0.124	0.445	677.782	17.893	3.344	0.000	10.043	0.000	0.155	1.506	0.001
rat99	5.843	0.167	1.064	803.023	20.747	4.19	0.056	10.816	0.000	0.098	4.873	0.004
kroA100	6.674	0.161	2.918	749.98	0.774	4.357	0.042	10.885	0.000	0.117	2.534	0.003
kroB100	8.232	0.169	0.732	750.806	18.792	4.393	0.263	10.716	0.000	0.174	0.585	0.003
kroC100	6.127	0.175	0.946	758.839	1.111	4.409	0.000	10.776	0.000	0.103	5.573	0.003
kroD100	5.637	0.168	1.049	754.058	11.206	4.338	0.000	10.762	0.000	0.127	3.851	0.004
kroE100	3.939	0.163	1.163	710.341	23.053	4.439	0.983	10.818	0.000	0.133	4.352	0.005
rd100	3.366	0.169	0.371	763.456	0.057	4.445	0.077	10.793	0.000	0.092	2.950	0.005
eil101	2.526	0.173	1.390	771.908	0.426	4.542	0.519	10.792	0.000	0.097	1.302	0.003
lin105	16.414	0.190	1.935	783.738	85.133	4.601	0.161	10.190	0.000	0.098	0.734	0.004
pr107	8.004	0.183	28.617	784.38	47.583	4.592	1.033	10.286	0.000	2.051	0.617	0.003
pr124	4.350	0.226	0.629	821.876	24.467	5.407	1.367	10.218	0.000	0.561	1.890	0.004
bier127	9.253	0.231	4.953	854.601	36.361	5.517	2.878	10.092	0.027	0.194	1.299	0.004
ch130	3.228	0.242	1.178	887.715	11.329	5.501	1.498	10.044	0.000	0.310	5.985	0.004
pr136	4.961	0.285	0.856	901.362	36.47	5.812	1.893	11.850	0.000	0.932	6.789	0.006
pr144	11.274	0.288	1.218	805.555	49.799	6.187	1.519	12.197	0.000	0.823	1.281	0.005
ch150	4.475	0.318	1.325	994.263	28.499	6.529	0.930	10.425	0.000	0.406	2.110	0.006
kroA150	12.380	0.318	2.396	964.421	31.413	6.479	2.174	10.112	0.000	0.397	7.097	0.006
kroB150	8.431	0.313	2.777	981.929	55.122	6.443	2.142	10.281	0.000	0.313	3.720	0.016
pr152	10.478	0.306	5.793	857.961	54.053	6.605	5.473	10.896	0.000	2.613	1.892	0.006
u159	8.799	0.331	0.823	982.689	37.904	6.952	0.207	10.398	0.000	0.165	5.962	0.283
rat195	16.879	0.478	5.340	1259.682	42.35	8.684	2.843	11.714	0.000	1.006	1.089	0.015
d198	434.654	0.474	31.723	1128.726	51.979	8.492	7.814	10.912	0.360	1.201	2.039	0.010
kroA200	15.661	0.495	6.224	1206.469	40.861	8.77	2.481	10.754	0.000	0.321	0.861	0.014
kroB200	18.612	0.477	5.606	1196.682	46.528	9.062	1.881	10.355	0.000	0.243	5.511	0.016
Mean	22.659	0.242	3.976	852.047	29.775	5.359	1.318	10.599	**0.013**	0.451	2.992	0.015

* The trials of the algorithm are currently set to 100.

In terms of computation time, DL-based end-to-end improvement algorithms demonstrate notably longer durations compared to other algorithm categories. For example, Costa et al.'s algorithm [63] necessitates over ten minutes to obtain high-quality solutions, while Xin et al.'s hybrid algorithm [71] can achieve solutions with a quality difference of merely 0.013% from the optimal solution within a mere 0.451 seconds.

As illustrated in Fig. 30, for real-world TSPs, hybrid algorithms, particularly Xin et al.'s [71], also demonstrate superior performance.

Overall, hybrid algorithms surpass other categories of algorithms in both solution quality and computation time.

5.3 Evaluation of generalization

We utilized models trained on small-scale TSPs by DL-based algorithms to solve larger-scale TSPs, aiming to assess the generalization performance of each typical algorithm.

The considered algorithms consist of those proposed by Kool et al. [50], Costa et al. [63], Joshi et al. [65], and Hudson et al. [66]. As Xin et al.'s algorithm [71] is not trained on fixed-scale TSPs, and Ye et al.'s algorithm [74] lacks a trained model, these two algorithms are not included in the tables presented in this section.

The test results reveal that with an increase in the scale of the problem, the performance of each algorithm typically declines, although to varying degrees across different algorithm categories. Subsequently, we will conduct a detailed analysis of the generalization performance of each algorithm category.

As shown in Table 4, when applying models trained on TSP20 instances to solve TSP50, TSP100, and TSP200 instances, Kool et al.'s algorithm [50], representing DL-based end-to-end construction algorithms, exhibits a substantial increase in the optimality gap. Specifically, it increases from 0.068% on TSP20 to 1.806%, 22.525%, and 66.220% for TSP50, TSP100, and TSP200 respectively, indicating a notable performance decline.

Similarly, Costa et al.'s algorithm [63], representing DL-based end-to-end improvement algorithms, shows a significant increase in the optimality gap from 0.001% on TSP20 to 2.482%, 23.181%, and 94.452% for TSP50, TSP100, and TSP200 respectively, also indicating a notable performance decline.

Early-stage hybrid algorithms, such as Joshi et al.'s algorithm [65], also experiences considerable performance declines, with optimality gap increasing from 0.035% on TSP20 to 28.892%, 66.655%, and 149.489% for TSP50, TSP100, and TSP200 respectively, further indicating notable performance declines.

In contrast, the later-stage hybrid algorithm proposed by Hudson et al. [66] show a much smaller performance decline, with the optimality gap increasing from 0.000% on TSP20 to 0.093%, 2.112%, and 6.365% for TSP50, TSP100, and TSP200 respectively, indicating a minor degradation in performance.

These findings underscore the superior generalization performance of the later-stage hybrid algorithm compared to DL-based end-to-end construction and improvement algorithms.

As shown in Table 5, when applying models trained on TSP50 instances to solve TSP100 and TSP200 instances, Kool et al.'s algorithm [50] exhibits a significant increase in optimality gap from 0.494% on TSP50 to 2.876% and 20.541% on TSP100 and TSP200, respectively, indicating considerable performance degradation.

Similarly, Costa et al.'s algorithm [63] experiences a relatively small rise in the optimality gap from 0.134% on TSP50 to 1.404% and 7.824% on TSP100 and TSP200, respectively, indicating a comparatively minor decline in performance.

Joshi et al.'s algorithm [65] shows a substantial increase in the optimality gap from 0.227% on TSP50 to 49.581% and 95.421% on TSP100 and TSP200, respectively, indicating significant performance degradation.

Hudson et al. [66] experiences a much smaller degradation in performance, with the optimality gap increasing from 0.002% on TSP50 to 1.028% and 5.765% on TSP100 and TSP200, respectively, making it the algorithm with the least

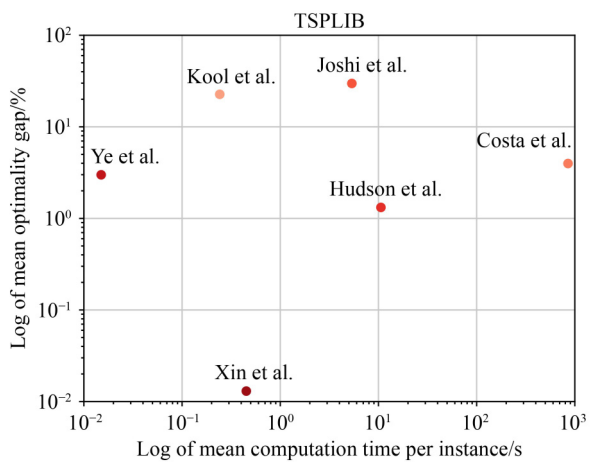

Fig. 30 Comprehensive perspective of solution quality and computation time for DL-based algorithms on real-world TSP instances from TSPLIB

Table 4 Generalization performance of models trained on TSP20 dataset for four representative DL-based algorithms

Algorithm	Category	Model trained on TSP20							
		TSP20		TSP50		TSP100		TSP200	
		Gap/%	Time/s	Gap/%	Time/s	Gap/%	Time/s	Gap/%	Time/s
Kool et al. [50]	C	0.068	0.026	1.806	0.062	22.525	0.158	66.220	0.480
Costa et al.* [63]	I	0.001	0.464	2.482	0.802	23.181	1.071	94.452	1.879
Joshi et al. [65]	D	0.035	0.974	28.892	9.176	66.655	13.264	149.489	21.563
Hudson et al. [66]	D	0.000	10.008	0.093	10.029	2.112	10.010	6.365	10.852

* The meaning of asterisk (*) is the same as that in Table 2.

Table 5 Generalization performance of models trained on TSP50 dataset for four representative DL-based algorithms

Algorithm	Category	Model trained on TSP50					
		TSP50		TSP100		TSP200	
		Gap/%	Time/s	Gap/%	Time/s	Gap/%	Time/s
Kool et al. [50]	C	0.494	0.065	2.876	0.160	20.541	0.480
Costa et al.* [63]	I	0.134	0.637	1.404	1.074	7.824	1.801
Joshi et al. [65]	D	0.227	2.261	49.581	13.197	95.421	21.637
Hudson et al. [66]	D	0.002	10.048	1.028	10.091	5.765	10.632

* The meaning of asterisk (*) is the same as that in Table 2.

deterioration in performance among these algorithms.

Similarly, it can be observed that the generalization performance of later-stage hybrid algorithms surpasses that of DL-based end-to-end construction and improvement algorithms.

As shown in Table 6, when models trained on TSP100 instances are used to solve TSP200 instances, the optimality gap of Kool et al.'s algorithm [50] exhibits a minor increase from 2.359% on TSP100 to 6.785%.

Conversely, Costa et al.'s algorithm [63] experiences a slight increase in the optimality gap from 0.758% on TSP100 to 2.785%.

The optimality gap of the early-stage hybrid algorithm by Joshi et al. [65] increases substantially from 1.516% on TSP100 to 49.477%, indicating a significant performance degradation.

In contrast, the later-stage hybrid algorithm by Hudson et al. [66] experiences a much smaller degradation in performance, with the optimality gap increasing from 0.454% on TSP100 to 3.285%, representing a minor performance degradation.

Comparatively, when models trained on TSP20 instances are used to solve TSP200 instances, the optimality gaps of algorithms proposed by Kool et al. [50], Costa et al. [63], Joshi et al. [65], and Hudson et al. [66] are 66.220%, 94.452%, 149.489%, and 6.365%, respectively. When models trained on TSP50 instances are applied to solve TSP200 instances, the optimality gaps of these algorithms are 20.541%, 7.824%, 95.421%, and 5.765%, respectively.

In summary, as the problem scale increases, the optimality gaps of algorithms proposed by Kool et al. [50], Costa et al. [63], and Joshi et al. [65] increase significantly, indicating a sharp decrease in solution quality with the increasing problem size. However, the later-stage hybrid algorithm proposed by Hudson et al. [66] shows a smaller decrease in solution quality when models trained on TSP20 are used to solve TSP50, TSP100, and TSP200 instances, demonstrating strong generalization performance. Overall, hybrid algorithms exhibit better generalization performance compared to DL-based end-to-end construction and improvement algorithms.

6 Conclusion and perspective

6.1 Conclusion

In summary, DL-based algorithms for solving TSP primarily fall into four categories: end-to-end construction algorithms, end-to-end improvement algorithms, direct hybrid algorithms, and LLM-based hybrid algorithms.

End-to-end construction algorithms employ neural networks to learn construction heuristics, constructing solutions from scratch, exhibiting rapid solving speed but often yield subpar solutions. Typically utilizing an encoder-decoder structure, the encoder generates embeddings for nodes, while the decoder computes the probability distribution of candidate nodes based on embeddings of them and previously selected nodes, gradually constructing complete solutions. Ptr-net are typical representatives, but due to the limitations of supervised learning in data annotation, the solution quality is often lower. Algorithms developed based on Ptr-net utilize reinforcement learning to train models, improving solution quality; however, a certain gap from the optimal solution remains.

End-to-end improvement algorithms leverage deep reinforcement learning to learn improvement heuristics, iteratively enhancing the quality of initial solutions. Similarly adopting an encoder-decoder structure, the decoder removes poor-quality edges and adds better-quality edges in each decoding step to refine the solution, iterating multiple times until obtaining higher-quality solutions. While these algorithms can achieve higher-quality solutions, they typically require longer computation times.

Direct hybrid algorithms combine DL directly with heuristic algorithms, leveraging the learning ability of DL and the search ability of heuristic algorithms, complementing each other and demonstrating stronger solving performance and generalization capability. Particularly, algorithms that integrate with heuristic methods such as GLS and LKH exhibit excellent performance for TSPs with different scales and distributions.

LLM-based hybrid algorithms have introduced novel

Table 6 Generalization performance of models trained on TSP100 dataset for four representative DL-based algorithms

Algorithm	Cateroty	Model trained on TSP100			
		TSP100		TSP200	
		Gap/%	Time/s	Gap/%	Time/s
Kool et al. [50]	C	2.359	0.167	6.785	0.479
Costa et al.* [63]	I	0.758	0.931	2.785	1.676
Joshi et al. [65]	D	1.516	4.327	49.477	21.592
Hudson et al. [66]	D	0.454	10.175	3.285	10.422

* The meaning of asterisk (*) is the same as that in Table 2.

approaches to heuristic design for TSP. They harness LLMs to automatically generate and refine heuristics, eliminating the need for model training. These algorithms leverage evolutionary algorithms to evolve construction or improvement heuristics, resulting in commendable performance. Given that this category of algorithms is still in its nascent stages of development, their performance may currently lag slightly behind that of direct hybrid algorithms. However, it is worth noting the potential for them to achieve even better performance in the future.

In conclusion, DL-based TSP solving algorithms continue to evolve, with various algorithms exhibiting pros and cons in solution quality, computation efficiency, and generalization capability. End-to-end construction algorithms offer fast speeds but lower solution quality, end-to-end improvement algorithms can achieve higher-quality solutions but require longer computation times, while direct hybrid algorithms have the potential to obtain very high-quality solutions in extremely short times, showing greater potential in overall performance. LLM-based hybrid algorithms provide novel avenues and possibilities for the automatic generation and refinement of effective algorithms. Moreover, within construction and hybrid algorithms, specialized algorithms designed for large-scale TSP have shown strong generalization abilities.

6.2 Perspective

One upcoming trend in TSP algorithm design involves integrating DL techniques to tackle larger and more intricate real-world instances. This includes developing neural network architectures capable of efficiently handling large-scale datasets and complex constraints, alongside advancements in training paradigms.

Another significant trend is further exploring the fusion of DL techniques with search methods to capitalize on their strengths, leading to more efficient and adaptable algorithms.

Moreover, leveraging large language model techniques will play a crucial role in uncovering underlying patterns and structures of TSP, facilitating the generation and refinement of heuristic algorithms for improved solution quality and computation efficiency.

Additionally, there is a growing emphasis on enhancing the interpretability and generalization of neural network models to enhance algorithm reliability and practicality. This involves clarifying the internal computational mechanisms of neural networks to improve user understanding and confidence in algorithmic decisions, as well as enhancing the generalization capabilities of neural networks across various data distributions and task environments to ensure algorithm stability and reliability in practical scenarios.

In particular, the traditional field of combinatorial optimization often focuses on worst-case scenarios, typically providing an error bound for algorithms. This presents a significant challenge for AI methods based on learning, which generally emphasize average-case performance and may lack guarantees for worst-case behavior. To address this, it is essential to integrate robust training methods, domain adaptation, and transfer learning into neural network models. Robust training methods can help neural networks become more resilient to variations in data, while domain adaptation and transfer learning can facilitate the application of learned models to new and diverse problem instances. Furthermore, combining theoretical error bounds with empirical performance metrics can provide a more comprehensive assessment of algorithm effectiveness, bridging the gap between worst-case guarantees and the strengths of learning-based approaches.

In summary, future research efforts will focus on leveraging different DL methodologies to address increasingly complex and large-scale TSP instances, while also advancing algorithm interpretability and generalization. These endeavors will inspire algorithm designers to create more effective algorithms for solving real-world problems, driving the advances of TSP algorithm design.

Acknowledgements This study was funded by the National Key R&D Program of China (2020YFA0907000), the National Natural Science Foundation of China (Grant Nos. 32270657, 32271297, 82130055, 62072435) and the Youth Innovation Promotion Association, Chinese Academy of Sciences. We appreciate the ComputeX center, ICT, CAS for providing computation service.

Competing interests The authors declare that they have no competing interests or financial conflicts to disclose.

References

1. Cook W J. In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton: Princeton University Press, 2012
2. La Maire B F J, Mladenov V M. Comparison of neural networks for solving the travelling salesman problem. In: Proceedings of the 11th Symposium on Neural Network Applications in Electrical Engineering. 2012, 21–24
3. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521(7553): 436−444
4. Wei K, Li T, Huang F, Chen J, He Z. Cancer classification with data augmentation based on generative adversarial networks. Frontiers of Computer Science, 2022, 16(2): 162601
5. Shi H, Wang J, Cheng J, Qi X, Ji H, Struchiner C J, Villela D A M, Karamov E V, Turgiev A S. Big data technology in infectious diseases modeling, simulation, and prediction after the COVID-19 outbreak. Intelligent Medicine, 2023, 3(2): 85−96
6. Wu Y, Zhang P, Shen H, Zhai H. Visualizing a neural network that develops quantum perturbation theory. Physical Review A, 2018, 98(1): 010701(R)
7. Zhang P, Shen H, Zhai H. Machine learning topological invariants with neural networks. Physical Review Letters, 2018, 120(6): 066401
8. Sun N, Yi J, Zhang P, Shen H, Zhai H. Deep learning topological invariants of band insulators. Physical Review B, 2018, 98(8): 085402

9. Jumper J, Evans R, Pritzel A, Green T, Figurnov M, Ronneberger O, Tunyasuvunakool K, Bates R, Žídek A, Potapenko A, Bridgland A, Meyer C, Kohl S A A, Ballard A J, Cowie A, Romera-Paredes B, Nikolov S, Jain R, Adler J, Back T, Petersen S, Reiman D, Clancy E, Zielinski M, Steinegger M, Pacholska M, Berghammer T, Bodenstein S, Silver D, Vinyals O, Senior A W, Kavukcuoglu K, Kohli P, Hassabis D. Highly accurate protein structure prediction with AlphaFold. Nature, 2021, 596(7873): 583−589
10. Tunyasuvunakool K, Adler J, Wu Z, Green T, Zielinski M, Žídek A, Bridgland A, Cowie A, Meyer C, Laydon A, Velankar S, Kleywegt G J, Bateman A, Evans R, Pritzel A, Figurnov M, Ronneberger O, Bates R, Kohl S A A, Potapenko A, Ballard A J, Romera-Paredes B, Nikolov S, Jain R, Clancy E, Reiman D, Petersen S, Senior A W, Kavukcuoglu K, Birney E, Kohli P, Jumper J, Hassabis D. Highly accurate protein structure prediction for the human proteome. Nature, 2021, 596(7873): 590−596
11. Baek M, DiMaio F, Anishchenko I, Dauparas J, Ovchinnikov S, Lee G R, Wang J, Cong Q, Kinch L N, Schaeffer R D, Millán C, Park H, Adams C, Glassman C R, Degiovanni A, Pereira J H, Rodrigues A V, Van Dijk A A, Ebrecht A C, Opperman D J, Sagmeister T, Buhlheller C, Pavkov-Keller T, Rathinaswamy M K, Dalwadi U, Yip C K, Burke J E, Garcia K C, Grishin N V, Adams P D, Read R J, Baker D. Accurate prediction of protein structures and interactions using a three-track neural network. Science, 2021, 373(6557): 871−876
12. Ju F, Zhu J, Shao B, Kong L, Liu T Y, Zheng W M, Bu D. CopulaNet: learning residue co-evolution directly from multiple sequence alignment for protein structure prediction. Nature Communications, 2021, 12(1): 2535
13. Davies A, Veličković P, Buesing L, Blackwell S, Zheng D, Tomašev N, Tanburn R, Battaglia P, Blundell C, Juhász A, Lackenby M, Williamson G, Hassabis D, Kohli P. Advancing mathematics by guiding human intuition with AI. Nature, 2021, 600(7887): 70−74
14. Fawzi A, Balog M, Huang A, Hubert T, Romera-Paredes B, Barekatain M, Novikov A, Ruiz F J R, Schrittwieser J, Swirszcz G, Silver D, Hassabis D, Kohli P. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature, 2022, 610(7930): 47−53
15. Tenachi W, Ibata R, Diakogiannis F I. Deep symbolic regression for physics guided by units constraints: toward the automated discovery of physical laws. The Astrophysical Journal, 2023, 959(2): 99
16. Romera-Paredes B, Barekatain M, Novikov A, Balog M, Kumar M P, Dupont E, Ruiz F J R, Ellenberg J S, Wang P, Fawzi O, Kohli P, Fawzi A. Mathematical discoveries from program search with large language models. Nature, 2024, 625(7995): 468−475
17. Wu N, Wang J, Zhao W X, Jin Y. Learning to effectively estimate the travel time for fastest route recommendation. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management. 2019, 1923–1932
18. Wu N, Zhao X W, Wang J, Pan D. Learning effective road network representation with hierarchical graph neural networks. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020, 6–14
19. Ji J, Wang J, Jiang Z, Ma J, Zhang H. Interpretable spatiotemporal deep learning model for traffic flow prediction based on potential energy fields. In: Proceedings of IEEE International Conference on Data Mining. 2020, 1076–1081
20. Wang J, Wu N, Zhao W X. Personalized route recommendation with neural network enhanced search algorithm. IEEE Transactions on Knowledge and Data Engineering, 2022, 34(12): 5910−5924
21. Wang Z, Pan Z, Chen S, Ji S, Yi X, Zhang J, Wang J, Gong Z, Li T, Zheng Y. Shortening passengers' travel time: a dynamic metro train scheduling approach using deep reinforcement learning. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(5): 5282−5295
22. Wang J, Ji J, Jiang Z, Sun L. Traffic flow prediction based on spatiotemporal potential energy fields. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(9): 9073−9087
23. Ji J, Wang J, Huang C, Wu J, Xu B, Wu Z, Zhang J, Zheng Y. Spatio-temporal self-supervised learning for traffic flow prediction. In: Proceedings of the 37th AAAI Conference on Artificial Intelligence. 2023, 4356–4364
24. Bengio Y, Lodi A, Prouvost A. Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research, 2021, 290(2): 405−421
25. Junior Mele U, Maria Gambardella L, Montemanni R. Machine learning approaches for the traveling salesman problem: a survey. In: Proceedings of the 8th International Conference on Industrial Engineering and Applications (Europe). 2021, 182–186
26. Shi Y, Zhang Y. The neural network methods for solving traveling salesman problem. Procedia Computer Science, 2022, 199: 681−686
27. Yang Y, Whinston A. A survey on reinforcement learning for combinatorial optimization. In: Proceedings of IEEE World Conference on Applied Intelligence and Computing. 2023, 131–136
28. Matai R, Singh S P, Mittal M L. Traveling salesman problem: an overview of applications, formulations, and solution approaches. In: Davendra D, ed. Traveling Salesman Problem, Theory and Applications. Rijeka: InTech, 2010, 1
29. Eastman W L. Linear programming with pattern constraints. Harvard University, Dissertation, 1958
30. Miller D L, Pekny J F. Exact solution of large asymmetric traveling salesman problems. Science, 1991, 251(4995): 754−761
31. Held M, Karp R M. A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics, 1962, 10(1): 196−210
32. Laporte G. The traveling salesman problem: an overview of exact and approximate algorithms. European Journal of Operational Research, 1992, 59(2): 231−247
33. Dantzig G, Fulkerson R, Johnson S. Solution of a large-scale traveling-salesman problem. Journal of the Operations Research Society of America, 1954, 2(4): 393−410
34. Miller C E, Tucker A W, Zemlin R A. Integer programming formulation of traveling salesman problems. Journal of the ACM, 1960, 7(4): 326−329
35. Gutin G, Punnen A P. The Traveling Salesman Problem and Its Variations. New York: Springer, 2007
36. Applegate D, Bixby R, Chvatal V, Cook W. Concorde TSP solver. See Math. uwaterloo. ca/tsp/concorde website, 2006
37. Rosenkrantz D J, Stearns R E, Lewis II P M. An analysis of several heuristics for the traveling salesman problem. SIAM Journal on Computing, 1977, 6(3): 563−581
38. Johnson D S, McGeoch L A. The traveling salesman problem: a case study in local optimization. Local Search in Combinatorial Optimization, 1997, 1(1): 215−310
39. Voudouris C, Tsang E. Guided local search and its application to the traveling salesman problem. European Journal of Operational Research, 1999, 113(2): 469−499
40. Nilsson C. Heuristics for the traveling salesman problem. Linkoping University, 2003, 38(0085-9): 26
41. Sui J, Ding S, Liu R, Xu L, Bu D. Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning. In: Proceedings of the 13th Asian Conference on Machine Learning. 2021, 1301–1316
42. Johnson D S. Local optimization and the traveling salesman problem. In: Proceedings of the 17th International Colloquium on Automata,

Languages, and Programming. 1990, 446–461
43. Helsgaun K. An effective implementation of the Lin-Kernighan traveling salesman heuristic. European Journal of Operational Research, 2000, 126(1): 106−130
44. Helsgaun K. General *k*-opt submoves for the Lin-Kernighan TSP heuristic. Mathematical Programming Computation, 2009, 1(2-3): 119−163
45. Helsgaun K. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems: technical report. Roskilde: Roskilde University, 2017, 966–980
46. Vinyals O, Fortunato M, Jaitly N. Pointer networks. In: Proceedings of the 28th International Conference on Neural Information Processing Systems. 2015, 2692–2700
47. Bello I, Pham H, Le Q V, Norouzi M, Bengio S. Neural combinatorial optimization with reinforcement learning. In: Proceedings of the 5th International Conference on Learning Representations. 2017
48. Nazari M, Oroojlooy A, Takáč M, Snyder L. Reinforcement learning for solving the vehicle routing problem. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. 2018, 9861–9871
49. Deudon M, Cournut P, Lacoste A, Adulyasak Y, Rousseau L M. Learning heuristics for the TSP by policy gradient. In: Proceedings of the 15th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. 2018, 170–181
50. Kool W, Van Hoof H, Welling M. Attention, learn to solve routing problems!2018, arXiv preprint arXiv: 1803.08475
51. Kwon Y D, Choo J, Kim B, Yoon I, Gwon Y, Min S. POMO: policy optimization with multiple optima for reinforcement learning. In: Proceedings of the 34th Conference on Neural Information Processing Systems. 2020, 21188–21198
52. Bresson X, Laurent T. The transformer network for the traveling salesman problem. 2021, arXiv preprint arXiv: 2103.03012
53. Pan X, Jin Y, Ding Y, Feng M, Zhao L, Song L, Bian J. H-TSP: hierarchically solving the large-scale traveling salesman problem. In: Proceedings of the 37th AAAI Conference on Artificial Intelligence. 2023, 9345–9353
54. Lischka A, Wu J, Basso R, Chehreghani M H, Kulcsár B. Less is more-on the importance of sparsification for transformers and graph neural networks for TSP. 2024, arXiv preprint arXiv: 2403.17159
55. Luo F, Lin X, Liu F, Zhang Q, Wang Z. Neural combinatorial optimization with heavy decoder: toward large scale generalization. In: Proceedings of the 37th Conference on Neural Information Processing Systems. 2024
56. Dai H, Khalil E B, Zhang Y, Dilkina B, Song L. Learning combinatorial optimization algorithms over graphs. In: Proceedings of the 31st Conference on Neural Information Processing Systems. 2017, 30
57. Ma Q, Ge S, He D, Thaker D, Drori I. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. 2019, arXiv preprint arXiv: 1911.04936
58. Drori I, Kharkar A, Sickinger W R, Kates B, Ma Q, Ge S, Dolev E, Dietrich B, Williamson D P, Udell M. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In: Proceedings of the 19th IEEE International Conference on Machine Learning and Applications. 2020, 19–24
59. Ouyang W, Wang Y, Weng P, Han S. Generalization in deep RL for TSP problems via equivariance and local search. 2021, arXiv preprint arXiv: 2110.03595
60. Sun Z, Yang Y. DIFUSCO: graph-based diffusion solvers for combinatorial optimization. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. 2023, 164
61. Liu H, Kuang Y, Wang J, Li X, Zhang Y, Wu F. Promoting generalization for exact solvers via adversarial instance augmentation. 2023, arXiv preprint arXiv: 2310.14161
62. Wu Y, Song W, Cao Z, Zhang J, Lim A. Learning improvement heuristics for solving routing problems. 2019, arXiv preprint arXiv: 1912.05784
63. Costa P R D O, Rhuggenaath J, Zhang Y, Akcay A. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In: Proceedings of the 12th Asian Conference on Machine Learning. 2020, 465–480
64. Ma Y, Li J, Cao Z, Song W, Zhang L, Chen Z, Tang J. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In: Proceedings of the 34th Conference on Neural Information Processing Systems. 2021, 11096–11107
65. Joshi C K, Laurent T, Bresson X. An efficient graph convolutional network technique for the travelling salesman problem. 2019, arXiv preprint arXiv: 1906.01227
66. Hudson B, Li Q, Malencia M, Prorok A. Graph neural network guided local search for the traveling salesperson problem. 2021, arXiv preprint arXiv: 2110.05291
67. Sui J, Ding S, Xia B, Liu R, Bu D. NeuralGLS: learning to guide local search with graph convolutional network for the traveling salesman problem. Neural Computing and Applications, 2024, 36(17): 9687−9706
68. Fu Z H, Qiu K B, Zha H. Generalize a small pre-trained model to arbitrarily large TSP instances. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence. 2021, 7474–7482
69. Sanyal S, Roy K. Neuro-Ising: accelerating large-scale traveling salesman problems via graph neural network guided localized Ising solvers. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(12): 5408−5420
70. Zheng J, He K, Zhou J, Jin Y, Li C M. Combining reinforcement learning with Lin-Kernighan-Helsgaun algorithm for the traveling salesman problem. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence. 2021, 12445–12452
71. Xin L, Song W, Cao Z, Zhang J. NeuroLKH: combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem. In: Proceedings of the 35th International Conference on Neural Information Processing Systems. 2021, 572
72. Zheng J, He K, Zhou J, Jin Y, Li C M. Reinforced Lin–Kernighan–Helsgaun algorithms for the traveling salesman problems. Knowledge-Based Systems, 2023, 260: 110144
73. Liu F, Tong X, Yuan M, Zhang Q. Algorithm evolution using large language model. 2023, arXiv preprint arXiv: 2311.15249
74. Ye H, Wang J, Cao Z, Berto F, Hua C, Kim H, Park J, Song G. Large language models as hyper-heuristics for combinatorial optimization. 2024, arXiv preprint arXiv: 2402.01145
75. Liu F, Tong X, Yuan M, Lin X, Luo F, Wang Z, Lu Z, Zhang Q. Evolution of heuristics: towards efficient automatic algorithm design using large language mode. 2024, arXiv preprint arXiv: 2401.02051
76. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. In: Proceedings of the 3rd International Conference on Learning Representations. 2015
77. Sato R. A survey on the expressive power of graph neural networks. 2020, arXiv preprint arXiv: 2003.04078
78. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017, 6000–6010
79. Naveed H, Khan A U, Qiu S, Saqib M, Anwar S, Usman M, Akhtar N, Barnes N, Mian A. A comprehensive overview of large language models. 2024, arXiv preprint arXiv: 2307.06435

80. Hu D. An introductory survey on attention mechanisms in NLP problems. In: Proceedings of Intelligent Systems Conference (IntelliSys) Volume 2. 2020, 432–448
81. Luong M T, Pham H, Manning C D. Effective approaches to attention-based neural machine translation. In: Proceedings of 2015 Conference on Empirical Methods in Natural Language Processing. 2015
82. Sperduti A, Starita A. Supervised neural networks for the classification of structures. IEEE Transactions on Neural Networks, 1997, 8(3): 714−735
83. Baskin I I, Palyulin V A, Zefirov N S. A neural device for searching direct correlations between structures and properties of chemical compounds. Journal of Chemical Information and Computer Sciences, 1997, 37(4): 715−721
84. Gori M, Monfardini G, Scarselli F. A new model for learning in graph domains. In: Proceedings of IEEE International Joint Conference on Neural Networks. 2005, 729–734
85. Scarselli F, Gori M, Tsoi A C, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE Transactions on Neural Networks, 2009, 20(1): 61−80
86. Duvenaud D, Maclaurin D, Aguilera-Iparraguirre J, Gómez-Bombarelli R, Hirzel T, Aspuru-Guzik A, Adams R P. Convolutional networks on graphs for learning molecular fingerprints. In: Proceedings of the 28th International Conference on Neural Information Processing Systems. 2015
87. Li Y, Tarlow D, Brockschmidt M, Zemel R S. Gated graph sequence neural networks. In: Proceedings of the 4th International Conference on Learning Representations. 2016
88. Dai H, Dai B, Song L. Discriminative embeddings of latent variable models for structured data. In: Proceedings of the 33rd International Conference on Machine Learning. 2016, 2702–2711
89. Gilmer J, Schoenholz S S, Riley P F, Vinyals O, Dahl G E. Neural message passing for quantum chemistry. In: Proceedings of the 34th International Conference on Machine Learning. 2017, 1263–1272
90. Hamilton W L, Ying R, Leskovec J. Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017, 1025–1035
91. Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th International Conference on Learning Representations. 2017
92. Bresson X, Laurent T. Residual gated graph convnets. 2017, arXiv preprint arXiv: 1711.07553
93. Velickovic P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph attention networks. In: Proceedings of the 6th International Conference on Learning Representations. 2018
94. Zhao W X, Zhou K, Li J, Tang T, Wang X, Hou Y, Min Y, Zhang B, Zhang J, Dong Z, Du Y, Yang C, Chen Y, Chen Z, Jiang J, Ren R, Li Y, Tang X, Liu Z, Liu P, Nie J Y, Wen J R. A survey of large language models. 2023, arXiv preprint arXiv: 2303.18223
95. Min B, Ross H, Sulem E, Veyseh A P B, Nguyen T H, Sainz O, Agirre E, Heinz I, Roth D. Recent advances in natural language processing via large pre-trained language models: a survey. ACM Computing Surveys, 2024, 56(2): 30
96. Tian H, Lu W, Li T O, Tang X, Cheung S C, Klein J, Bissyandé T F. Is ChatGPT the ultimate programming assistant-how far is it? 2023, arXiv preprint arXiv: 2304.11938
97. Jablonka K M, Schwaller P, Ortega-Guerrero A, Smit B. Is GPT-3 all you need for low-data discovery in chemistry? See Chemrxiv.org/engage/chemrxiv/article-details/63eb5a669da0bc6b33e97a35 website, 2023
98. Lee P, Bubeck S, Petro J. Benefits, limits, and risks of GPT-4 as an AI chatbot for medicine. New England Journal of Medicine, 2023, 388(13): 1233−1239
99. Nori H, King N, McKinney S M, Carignan D, Horvitz E. Capabilities of GPT-4 on medical challenge problems. 2023, arXiv preprint arXiv: 2303.13375
100. Cheng K, Guo Q, He Y, Lu Y, Gu S, Wu H. Exploring the potential of GPT-4 in biomedical engineering: the dawn of a new era. Annals of Biomedical Engineering, 2023, 51(8): 1645−1653
101. Blocklove J, Garg S, Karri R, Pearce H. Chip-chat: challenges and opportunities in conversational hardware design. In: Proceedings of the 5th ACM/IEEE Workshop on Machine Learning for CAD. 2023, 1–6
102. He Z, Wu H, Zhang X, Yao X, Zheng S, Zheng H, Yu B. ChatEDA: a large language model powered autonomous agent for EDA. 2023, arXiv preprint arXiv: 2308.10204
103. Shah D, Equi M R, Osiński B, Xia F, Ichter B, Levine S. Navigation with large language models: semantic guesswork as a heuristic for planning. In: Proceedings of the 7th Conference on Robot Learning. 2023
104. Xiao H, Wang P. LLM A*: human in the loop large language models enabled A* search for robotics. 2023, arXiv preprint arXiv: 2312.01797
105. Yang C, Wang X, Lu Y, Liu H, Le Q V, Zhou D, Chen X. Large language models as optimizers. In: Proceedings of ICLR 2024 Conference. 2024
106. Wu X, Zhong Y, Wu J, Jiang B, Tan K C. Large language model-enhanced algorithm selection: towards comprehensive algorithm representation. In: Proceedings of the 33rd International Joint Conference on Artificial Intelligence. 2024
107. Liu F, Lin X, Wang Z, Yao S, Tong X, Yuan M, Zhang Q. Large language model for multi-objective evolutionary optimization. 2024, arXiv preprint arXiv: 2310.12541
108. Sutskever I, Vinyals O, Le Q V. Sequence to sequence learning with neural networks. In: Proceedings of the 27th International Conference on Neural Information Processing Systems. 2014, 3104–3112
109. Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors. Nature, 1986, 323(6088): 533−536
110. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735−1780
111. Mnih V, Badia A P, Mirza M, Graves A, Harley T, Lillicrap T P, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd International Conference on Machine Learning. 2016, 1928–1937
112. Christofides N. Worst-case analysis of a new heuristic for the travelling salesman problem. Pittsburgh: Carnegie-Mellon University, 1976
113. Perron L, Furnon V. OR-Tools. See developers.google.com/optimization/ website. 2023
114. Williams R J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 1992, 8(3-4): 229−256
115. Kingma D P, Ba J. Adam: a method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations. 2015
116. Sutton R S, Barto A G. Reinforcement learning: an introduction. Robotica, 1999, 17(2): 229−235
117. Reinelt G. TSPLIB—A traveling salesman problem library. ORSA Journal on Computing, 1991, 3(4): 376−384
118. Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. 2020, 574
119. Perez E, Strub F, De Vries H, Dumoulin V, Courville A. FiLM: visual reasoning with a general conditioning layer. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence. 2018
120. Misevičius A. Using iterated Tabu search for the traveling salesman

problem. Information Technology and Control, 2004, 32(3): 29−40
121. Vikhar P A. Evolutionary algorithms: a critical review and its future prospects. In: Proceedings of International Conference on Global Trends in Signal Processing, Information Computing and Communication. 2016, 261–265
122. Arnold F, Sörensen K. Knowledge-guided local search for the vehicle routing problem. Computers & Operations Research, 2019, 105: 32−46

Jingyan Sui is a PhD candidate at State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, China. Her primary research interests encompass algorithm design, machine learning, deep learning, and combinatorial optimization.

Shizhe Ding is a PhD candidate at State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, China. His primary research interests encompass machine learning, and combinatorial optimization.

Xulin Huang is a graduate student at Henan Institute of Advanced Technology, Zhengzhou University, and State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, China. His primary research interests encompass algorithm design, deep learning, combinatorial optimization, and bioinformatics.

Yue Yu is a graduate student at Hangzhou Institute for Advanced Study, and State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, China. Her main research interests encompass algorithm design, deep learning, and bioinformatics.

Ruizhi Liu is a PhD candidate at State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, China. His main research interests encompass algorithm design, deep learning, combinatorial optimization, and chip design.

Boyang Xia is a graduate student at State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, China. His main research interests encompass machine learning and combinatorial optimization.

Zhenxin Ding is a graduate student at State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, China. His main research interests encompass algorithm design, machine learning, and combinatorial optimization.

Liming Xu is a graduate student at State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, China. Her main research interests encompass algorithm design, machine learning, and combinatorial optimization.

Haicang Zhang is a PhD, an associate researcher, a Master's supervisor at State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, China. His main research focuses on machine learning, protein design, and protein structure prediction.

Chungong Yu is a Master, a senior engineer at State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, China. His main research interests encompass bioinfomatics and protein structure prediction.

Dongbo Bu is a PhD, a Professor, and a PhD supervisor at State Key Lab of Processor, Institute of Computing Technology, Chinese Academy of Sciences, University of Chinese Academy of Sciences, and Central China Institute of Artificial Intelligence, China. His main research interests encompass algorithm design, bioinformatics, protein structure prediction, and deep learning.