Faster Capacitated Arc Routing: A Sequence-to-Sequence Approach

WENJING HONG\(^{1,2}\), (Member, IEEE), AND TONGLIN LIU\(^{3}\)

\(^{1}\)Guangdong Provincial Key Laboratory of Brain-Inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China

\(^{2}\)Guangdong-Hong Kong-Macao Greater Bay Area Center for Brain Science and Brain-Inspired Intelligence, Guangzhou 510515, China

\(^{3}\)Science and Technology on Complex System Control and Intelligent Agent Cooperation Laboratory, Beijing Electro-Mechanical Engineering Institute, Beijing 100074, China

Corresponding authors: Tonglin Liu (ltl2006@126.com) and Wenjing Hong (hongwj@sustech.edu.cn)

This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2018B030340001, and in part by the National Natural Science Foundation of China under Grant 62106098 and Grant 61703382.

ABSTRACT The Capacitated Arc Routing Problem (CARP) is an NP-hard optimization problem that has been investigated for decades. Heuristic search methods are commonly used to solve it. However, given a CARP instance, most heuristic search algorithms require plenty of time to iteratively search for the solution from scratch, and hence may be impractical for emerging applications that need a solution to be obtained in a very short time period. In this work, a novel approach to efficiently solve CARP is presented. The proposed approach replaces the heuristic search process with the inference phase of a trained Deep Neural Network (DNN), which is trained to take a CARP instance as the input and outputs a solution to the instance. In this way, CARP could be solved by a direct mapping rather than by iterative search, and hence could be more efficient and more easily accelerated by the use of GPUs. Empirical study shows that the DNN-based solver can achieve significant speed-up with minor performance loss, and up to hundreds of times acceleration in extreme cases.

INDEX TERMS Capacitated arc routing problem, sequence-to-sequence, deep neural network.

I. INTRODUCTION

The Capacitated Arc Routing Problem (CARP), with numerous practical applications such as garbage collection and post-delivery [1] is an important NP-hard combinatorial optimization problem that has attracted researchers for decades of investigations [2]–[5]. Recently, emerging applications, such as autonomous driving systems, require almost real-time (online) solving of CARP instances. However, despite the excellent performance in terms of solution quality, the state-of-the-art CARP solvers are not computationally efficient enough. The main reason is that these approaches are mostly heuristic search algorithms that typically start from scratch to tackle a CARP problem through a trial-and-error procedure [6], [7]. Thus, they could neither be easily parallelized nor leverage on experience to rapidly achieve a sufficiently good solution.

Motivated by the fast development of deep learning techniques [8]–[11], a new potential paradigm for solving CARP is to build models with machine learning techniques, rather than through manually designing heuristic search algorithms. Specifically, for a given problem class, one can utilize a set of problem instances, for which a target solution (e.g., the optimal solution or a solution that is good enough) is known, to train models that act as solvers to an optimization problem. When a new problem instance arrives, the model can directly generate a solution to the instance.

The above paradigm offers advantages over heuristic search algorithms in two aspects. First, solving a CARP instance is reformulated as the inference procedure of a learning model, which could be much more easily accelerated with the aid of new computing facilities [12], [13], e.g., GPUs. Second, the training process of the learning model involves the information of other instances, the solution of which could be first labeled (in an offline manner) by the state-of-the-art search algorithms. Hence, the training process could be viewed as transferring the knowledge inside the heuristic solver into the learned model. In this way, good solutions could be generated for a new problem instance even
faster, since it leverages on previously solved instances rather than start from scratch. Besides, an algorithm under this paradigm can essentially be considered as a meta-heuristic that provides an alternative solution to legacy heuristic approaches. In terms of solution quality, such a machine learning approach, might not perform as well as state-of-the-art search algorithms, since that depends heavily on the training data and training algorithm. However, it may offer a better trade-off between solution quality and computational efficiency for its high inference speed.

In this paper, a deep learning approach to CARP is proposed following the above idea. Specifically, to an input CARP instance, the proposed method, namely Deep Arc Routing Solver (DARS), has four steps. The first is the pre-sorting step. Task edges in the instance are sorted as a sequence of edges where the distance between adjacent edges in the sequence will be smaller. Then, it produces a corresponding numerical representation for each edge in the sorted sequence. This is done by utilizing a graph embedding technique, i.e., node2vector [14]. Now, the representation of task edges is actually a sequence of numerical edge vectors. In the third step, the edge sequence is fed into a Recurrent Neural Network (RNN) based sequence-to-sequence model [15], called Pointer Network (Ptr-Net), to generate the output solution for the input instance. It should be noted here a masking scheme is utilized during the generation. Finally, post-sorting is used to further improve the solution quality.

To summarize, this work contributes to the literature in two aspects. First, a deep neural network-based DARS is proposed for CARP, and it achieves significantly faster speed than heuristic search methods with the help of GPUs. Second, DARS is a new approach that formulates a combinatorial optimization problem as a sequence-to-sequence prediction problem. Empirical results on CARP indicate this formulation achieves better solution quality than other attempts of applying machine learning to combinatorial optimization.

The rest of the paper is organized as follows. Section II presents the related work. Section III introduces the proposed method. Section IV presents the empirical studies. Section V concludes the paper.

II. RELATED WORK AND NOTATIONS

A. RELATED WORK

In the literature, the state-of-the-art solvers for CARP are mostly heuristic search algorithms [6], [7], [16]. In general, these methods all follow a trial-and-error paradigm, i.e., candidate solutions are iteratively generated and tested, until a predefined termination condition is met. The best solution found so far is adopted. Studies along this line, e.g., the Memetic Algorithm with Extended Neighborhood Search (MAENS) [7] and [16], mainly focus on designing search operators to generate high-quality solutions more effectively, and have been reported to achieve the best performance on many benchmark test instances.

More recently, inspired by the achievements of deep learning, several attempts have been made to tackle NP-hard combinatorial optimization problems with deep learning. These meta-heuristics may provide a sufficiently good solution to such problems with less computational effort than iterative methods or simple heuristics. The first work is called Pointer Network (Ptr-Net) [15] which mainly follows the encoder-decoder architecture [17] with a customized pointer attention mechanism. It is trained in a supervised manner. With the same problem formulation proposed by Ptr-Net, various deep neural solvers have arisen with different architectures or different training modes in these years.

Some solvers are trained by reinforcement learning. The work of [18] adopts a modified version of Ptr-Nets trained by the policy gradient algorithm. The work of [19] replaces the encoder of Ptr-Nets with a set of graph embedding and is applied to the Vehicle Routing Problem (VRP). The work of [20] proposes a new encoder-decoder architecture inspired by Transformer [21] with a new attention mechanism different from Ptr-Net. It uses REINFORCE [22] to train the solver. The work of [23] constructs a solution by adding nodes incrementally. It utilizes a graph embedding model, called structure2vec [24], to parameterize the Q-value of nodes. It is trained by the Q-learning algorithm. A neural network based solver optimized by reinforcement learning which is a combination of a modified graph convolutional network and two encoder-decoder models is presented in [25].

Other works may use supervised training. The work of [26] employs recurrent neural networks to solve quadratic assignment problem. The model incorporates a so-called objective-based learning. This learning method only back-propagates the gradients if the predicted solution yields a worse objective than the approximate solution. The work of [27] uses graph neural networks for decision variant of the Traveling Salesman Problem (TSP). It is trained with approximate solutions generated by Concorde TSP solver.

Although this work is inspired by the above deep learning approaches, it is distinct from previous studies in three aspects. First, concerning the specific target problem, no previous study addresses CARP. Although CARP is similar to TSP and VRP, it is in essence a more challenging problem. To be specific, CARP can be viewed as a TSP with an additional capacity constraint, and a CARP instance with $k$ edges can be converted into a capacitated VRP with $3k + 1$ nodes theoretically [28]. Second, CARP is a specific permutation-based optimization problem [29]. Previous deep learning approaches formulate such type of problem as a set-to-sequence prediction problem, i.e., sequence of input elements for Deep Neural Networks (DNNs) is not considered in their formulation, while this work formulates the problem as a sequence-to-sequence prediction problem. Finally, previous studies emphasize more on the solution quality of the trained model, while the potential speed-up of DNNs in comparison to state-of-the-art heuristic search algorithms is the focus of this work.
B. NOTATIONS OF CARP

This work considers the CARP on a graph $G(V, E, A)$, where $V$ is the node set, $E$ is the edge set and $A$ is the arc set. For $v e \in E$, it relates to a pair of arcs $(\vec{e}, \bar{e})$ which represents opposite directions. Note that the direction of $\vec{e}$ is arbitrarily defined. The starting and the ending vertices of the edge $\vec{e}$ are denoted as $u_1(\vec{e})$ and $u_2(\vec{e})$, respectively. There are three properties assigned to $e$, i.e., demand, serving cost and deadheading cost. They are denoted of which represents opposite directions. Note that the direction of $\vec{e}$ is arbitrarily defined. The starting and the ending vertices of the edge $\vec{e}$ are denoted as $u_1(\vec{e})$ and $u_2(\vec{e})$, respectively. There are three properties assigned to $e$, i.e., demand, serving cost and deadheading cost. They are denoted as $d(e)$, $sc(e)$ and $dc(u_1(\vec{e}), u_2(\vec{e}))$, respectively. If $d(e) > 0$ then $sc(e) > 0$, which indicates $e$ is a task edge to serve; yet, if $d(e) = 0$ then $sc(e) = 0$ and $e$ contains no tasks. Additionally, the deadheading cost of passing the edge $e$ in either direction without serving the task on $e$ is expressed as $dc(u_1(\vec{e}), u_2(\vec{e}))$ or $dc(u_1(\bar{e}), u_2(\bar{e}))$, and it has $dc(u_1(\vec{e}), u_2(\bar{e})) = dc(u_1(\bar{e}), u_2(\vec{e}))$. It should be noted that vehicles are all located in a special node $v_0 \in V$, called depot. A view of graph is shown in the top-left corner of Fig. 1.

Taking $E$ as input, a solution of CARP is a set of routes, denoted by $S = [R_1, R_2, \cdots, R_m]$ where $m$ is the number of routes. For $\forall a_i \in A$, $1 \leq i \leq m$, $1 \leq j \leq |R_i|$, the $j$-th route contains a sequence of arcs belonging to task edges and is expressed as $[R_i]$, where $R_i = (a_{i1}, a_{i2}, \cdots, a_{i|R_i|})$, $d(a_{ij}) \geq 0$, $j$ is the order of tasks to be served in $R_i$ and $|R_i|$ is the number of arcs in $R_i$. A valid $S$ must satisfy 3 constraints:

1) Each route should start and end at the depot $v_0$.
2) Each task is served in only one route (but it can be traversed without serving for unlimited times).
3) The total demand of a route should not be larger than the vehicle’s capacity $Q$.

Note that the $S$ can be equally represented as a sequence

$$S = (v_0, a_{11}, \cdots, a_{1|R_1|}, v_0, \cdots, v_0, a_{m1}, \cdots, a_{m|R_m|}, v_0). \quad (1)$$

according to the constraint 1).

The cost of a route $R_i$ can be calculated as:

$$C(R_i) = \sum_{j=1}^{|R_i|} sc(a_{ij}) + \sum_{j=2}^{\infty} dc(u_2(a_{(j-1)}), u_1(a_{j}))$$

$$+ dc(v_0, u_1(a_{11}')) + dc(u_2(a_{|R_i|}), v_0). \quad (2)$$

where $dc(a, b)$ for node $a \in V$ and $b \in V$ denotes the total deadheading cost of the shortest path between node $a$ and $b$, and the path is found by Dijkstra algorithm [30]. The objective of CARP is finding a solution $S^*$ that minimizes the total cost $\sum_{r \epsilon S^*} C(r)$ while satisfying the aforementioned constraints.

III. DARS FOR CARP

In this section, we first model the CARP solving as a prediction task which maps an input sequence to an output sequence. Then, based on the formulation, the overall structure of the proposed solver is illustrated. Notations in Section II-B is the basics for this section.

A. FORMULATION

For a set of $n$ CARP instances $E = \{E_1, E_2, \cdots, E_n\}$, by applying a heuristic algorithm, a corresponding output solution set $\{\hat{S}_1, \hat{S}_2, \cdots, \hat{S}_n\}$ is generated and further a data set denoted as $D = \{(E_1, \hat{S}_1), (E_2, \hat{S}_2), \cdots, (E_n, \hat{S}_n)\}$ is constructed. Given a pair $(E_k, \hat{S}_k) \in D$, the process of CARP solving can be reformulated as a sequence to sequence prediction task.

It first sorts the edges in $E_k$ to ensure that the distance between adjacent elements in sequence $\hat{E}_k$ is minimized. This step called presorting is a greedy method. Then the problem become predicting the sequence $\hat{S}_k$ based on a sequence $\hat{E}_k$. This is a sequence-to-sequence prediction task and the conditional probability $p(\hat{S}_k | \hat{E}_k; \theta)$ can be estimated using the probability chain rule with respect to (1), i.e.,

$$p(\hat{S}_k | \hat{E}_k; \theta) = \prod_{i=1}^{m} p_0(R_i | R_1, \cdots, R_{i-1}, \hat{E}_k; \theta)$$

$$= \prod_{i=1}^{m} p_0(a_i | v_0, a_1, a_2, \cdots, a_{i-1}, \hat{E}_k; \theta). \quad (3)$$

Here, $p_0(a_i | v_0, a_1, a_2, \cdots, a_{i-1}, \hat{E}_k; \theta)$ is computed by an RNN model with parameters $\theta$ and the best $\theta^*$ can be learned by the maximum likelihood estimation over the training set $D$, i.e.,

$$\theta^* = \arg \max_\theta \sum_{k=1}^{n} \log p(\hat{S}_k | \hat{E}_k; \theta), \quad (4)$$

using optimization methods like stochastic gradient decent in a supervised training mode.

B. THE PROPOSED SOLVER

Based on (3) (4), a new solver consists of edge pre-sorting, graph embedding, DNN with masking and post-sorting is proposed.

Given a graph $G(V, E, A)$, the input is the set of task edges and it is denoted as $E' = \{e_1, e_2, \cdots, e_T\}$, where $d(e_i) > 0$ and $T$ is the number of tasks. The solver will generate a sequence of input elements, denoted as $S$. The illustration of the whole pipeline is shown in Fig. 1. In the following, the details of the method are explained.

1) PRE-SORTING IN GREEDY

$E'$ is a set of edges which presents no information about the distances among edges. However, for CARP, if two task edges are very close to each other in reality, they are likely to appear in the same route of the output solution. This is because that once two task edges become closer in route $R_i$, it will lead to a better value of $C(R_i)$ in (2) and makes $R_i$ more likely to be added into the final solutions.

To address this drawback, a sequence $\bar{E}$ is generated by sorting the elements in $E'$ with respect to minimizing the distance between adjacent elements. Here, the distance $d\bar{e}(e_1, e_2)$ between edges $\{e_1, e_2\} \in E'$ is defined as the average deadheading cost among endpoints on different
edges \([31]\),
\[
dhc(e_1, e_2) = \frac{1}{4} \left( \text{d}(u_1, e_1^1) + \text{d}(u_1, e_2^1) + \text{d}(u_2, e_1^2) + \text{d}(u_2, e_2^2) \right),
\]
and the distance \(dhe(v_0, e)\) between the depot \(v_0\) and an edge \(e\) is defined as:
\[
dhe(v_0, e) = \frac{1}{2} \left( \text{d}(v_0, u_1(e^1)) + \text{d}(v_0, u_2(e^2)) \right).
\]
Elements in \(E'\) are sorted in a greedy manner, as shown in Algorithm 1. Specifically, it starts from the depot \(v_0\) (line 2) to select the nearest task edge as the first element in \(\tilde{E}\). Then, it iteratively selects the next nearest element by measuring the distances between unselected edges and the current selected one (line 4). The sorting is finished until all the elements in \(E'\) are selected. Finally, the sequence \(\tilde{E} = (e_1^1, e_2^1, \ldots, e_T^1)\) is derived.

2) GRAPH REPRESENTATION USING EMBEDDING

DNN accepts numerical vectors as input and cannot process a graph directly, so that a graph embedding method, i.e., node2vec \([14]\) is employed. Node2vec computes a \(l\)-dimensional embedding \(emb(v) \in \mathbb{R}^l\) for each node \(v \in V\) and \(l \ll |V|\). The adjacency information can be preserved by the embedding of all nodes.

Moreover, the information graph of the CARP instance contains not only the adjacency information of graph structure but also other key information like the depot, the demands on the edges, and the vehicle capacities. These necessary features cannot be represented by \(emb(v)\). Thus, the numerical representation \(\mu_e\) for each \(e \in \tilde{E}\) used in this paper is expressed as follows:
\[
\mu_e = (emb(u_1(e^1)), emb(u_2(e^1)), f(u_1(e^1)), f(u_2(e^1)), d(e), \frac{d(e)}{Q}) \in \mathbb{R}^{2l+4}
\]
where for a node \(v \in V\), \(f(v) = 1\) if \(v = v_0\), otherwise \(f(v) = 0\). Note that it can use either \(e^1\) or \(e^2\) to generate \(\mu_e\) for \(e\), because the edges are undirected which is different from

Algorithm 1: Pre-Sorting Algorithm

**Input:** A set of task \(E' = \{e_1, e_2, \ldots, e_T\}\)

**Output:** A sequence of tasks

1. Let \(\tilde{E}\) be an empty sequence;
2. Let \(t = v_0\);
3. while \(E' \neq \emptyset\) do
   4. Select \(e'\) from \(E'\) with minimal \(dhe(v_0, e')\), \(\forall e' \in E'\);
   5. Insert \(e'\) into the tail of \(\tilde{E}\);
   6. \(t = e'\);
   7. \(E' = E' \setminus \{e'\}\);
4. end
9. return \(\tilde{E}\)
Therefore, the representation of \( \bar{E} \) becomes \( \bar{E}_{emb} \). It is expressed as follows:

\[
\bar{E}_{emb} = (\mu\bar{e}_1, \mu\bar{e}_2, \ldots, \mu\bar{e}_T).
\]

(8)

3) POINTER-NETWORK

In this work, Ptr-Net with trainable parameters \( \theta \) is employed to estimate \( p_\theta(\bar{a}_t^1 \ldots \bar{a}_T; \theta) \) and \( p(\bar{E}_{emb}; \theta) \) with respect to (3). Ptr-Net is an encoder-decoder model which is capable to process sequential inputs for combinatorial optimization problem while existing applications [15], [18], [19] only use it to process non-sequential data. As indicated in Section III-A, this work transforms the CARP solving to a sequence-to-sequence prediction task, therefore Ptr-Net is suitable. Additionally, it is worth noting that the purpose of this work is not to propose a new sequence-to-sequence learning model, but to use a sequence-to-sequence learning model to assist CARP solving and to verify that the use of sequence-to-sequence learning can help speed up the solving. There are many well-established models that are available to address the resultant task [32]–[34] and it is believed that the performance may be further improved with more advanced methods.

As illustrated in Fig. 1, for a CARP instance, the encoder network reads the elements of \( \bar{E}_{emb} \) one by one and produces a hidden state sequence \( h = (h_{e_1}, h_{e_2}, \ldots, h_{e_T}) \). For \( i \leq T \), \( h_{e_i} \) is the corresponding encoder hidden state of \( \mu\bar{e}_i \) which takes \( \mu\bar{e}_i \) and \( h_{e_{i-1}} \) as input. Particularly, For \( i = 1 \), \( h_{e_0} \) is an initial hidden state. After all the input elements are processed, the decoder begins generating outputs. At each decoding step \( t \), the decoder retrieve the input considering its output at step \( t - 1 \) and produces a hidden state \( h'_t \).

A pointer attention mechanism receives \( h'_t \) and \( h \) to generate a distribution over elements in the input sequence \( \bar{E} \). The output of the decoder at step \( t \) is an index of elements and is determined by the distribution.

4) MASKING SCHEME AND POST-SORTING IN TESTING

As presented above, Ptr-Net selects an output element according to a distribution over the elements of \( \bar{E} \) at each decoding step. That means there are about \( O(T^2) \) choices to explore for the best result. To deal with this, a masking scheme which is similar to [19], is adopted in testing. Note that masking is only used in testing, because the best output element in each decoding step is known in training.

In each decoding step, two types of edges will be masked: the edges that are severed and the edges whose demands are larger than the current load of the vehicle. Masked edges would not be considered in this step, and the edge with the highest probability in the unmasked edges are output.

Post-sorting is adopted to further improve the quality of the solution. This is another application of the idea appeared in pre-sorting. To be specific, for each route in the solution, the order of tasks is first organized by the presorting method described in Algorithm 1. Then to each arc in the final solution, its direction is tuned by fixing the starting vertex as the endpoint which is closer to the previous edge in the sequence.

5) SUMMARY

Given a graph \( G(V, E, A) \), the proposed solver first extracts the task edges as input \( E' \). Then the sequence \( \bar{E} \) is generated by adopting Algorithm 1 to sort the elements in \( E' \) (i.e., step 1 in Fig. 1). As \( \bar{E} \) cannot be processed by DNN, the graph embedding is employed to generate the numerical representation \( \bar{E}_{emb} \) for \( \bar{E} \) (i.e., step 2 in Fig. 1). The \( \bar{E}_{emb} \) is subsequently fed into Ptr-Net and Ptr-Net generates the elements of the output sequence step by step (i.e., step 3 in Fig. 1). During the generation masking scheme is used to filter out the invalid solutions (i.e., step 4 in Fig. 1). Finally, the output sequence is determined by fine-tuning the sequence and direction of arcs in the solution by post-sorting (i.e., step 5 in Fig. 1).

IV. EXPERIMENTS

In following experiments, MAENS [7], a well-known CARP solver is used as the heuristic method which DARS should learn from. VRP-RL [19] is a neural solver for VPR which is trained by reinforcement learning, and is modified to solve CARP. The modification is simply using the (8) as the data repre-sentation and determining the directions of output arcs by that used in posting-sorting.

The official implementation of MAENS in C++ is used in experiments and other algorithms are based on PyTorch. PyTorch is a machine learning library for Python and can utilizes GPUs to accelerate DNNs. All experiments are run on a workstation with two intel CPUs E5-2678 v3@2.50GHz and one NVIDIA GTX 1070.

A. DATASETS AND CONFIGURATIONS

The sizes of data sets used in previous CARP research are too small for training DNN. Thus, similar to the work of [25], we synthesize three new CARP data sets, named BJ-30, BJ-50, BJ-100 and BJ-150, with around 30, 50, 100 and 150 edges, respectively. All of the four data sets are generated from the roadmap of Beijing in China (https://www.openstreetmap.org). They are the set of \( E \) for \( D \) defined in Section III-A.

The roadmap includes 409,650 nodes and 778,304 undirected edges. Each dataset is first generated by sampling a number of instances from this roadmap and then divided into two disjoint subsets with specified proportions. One of these two subsets is used as the training set and the other is used as the test set. More specifically, each instance is randomly sampled from this roadmap using the same sampling settings. That is, for each instance, the graph is extracted from the roadmap randomly; the depot is determined randomly and the edges are randomly selected as tasks. Other features like the values of vehicle capacity and task demands are also generated randomly within the specific ranges. An illustration of sampling an instance is shown in Fig. 2. Note that instances that cannot be solved by MAENS will be regenerated until
each instance in data has a corresponding solution from MAENS. In this work, each dataset is divided into a training set with 1 million instances and a test set with 10,000 instances. The information of datasets is shown in Table 1.

The embedding size \( l \) for \( \text{emb}(\cdot) \) is 4. The Ptr-Net in DARS has 2 GRU layers 128 hidden units in each layer. The Ptr-Net is trained with Adam optimization algorithm [35] and the initial learning rate is set to \( 5 \times 10^{-3} \). Each batch size is 256 and the training lasts for 200 epochs. The weights of Ptr-Net are initialized with the orthogonal initialization [36] and the L2 norm of gradients is clipped to 2.0 to prompt generalization ability.

For convenience, we call the solutions generated by MAENS as the reference solutions, denoted as \( S^* \), and the solutions generated by the proposed method, denoted as \( S' \). For a solution \( S_i \in S' \), \( S_i^* \in S^* \) is the corresponding reference solution. All the feasible solutions (i.e., they satisfy the constraints in Section II-B) in \( S' \) construct a set \( \hat{S} = \{ S_i | \forall i < n, S_i \text{ is valid}, S_i \in S' \} \).

To evaluate the performance of different methods, the commonly used metric Mean Percentage Error (MPE) is employed, i.e.,

\[
MPE = \frac{100\%}{|\hat{S}|} \sum_{S_i \in \hat{S}} \frac{\sum_{r \in S_i^*} C(r) - \sum_{r \in S_i} C(r)}{\sum_{r \in S_i} C(r)}. \tag{9}
\]

Note that MPE defined here is meaningless for MAENS, since there are no references solutions for MAENS.

B. RESULTS AND DISCUSSIONS

1) SOLUTION QUALITY OF DARS

Table 2 show the comparison results in terms of solution quality for MAENS, VRP-RL and DARS. In order to thoroughly investigate the effects of different components of DARS, based the trained DARS model, we delete different modules to examine the changes on performance. The resultant models are referred to as DARS1/2/3/4, respectively, and their configurations are shown in Table 2.

According to the results in Table 2, VRP-RL performs worse than DARS. Moreover, as the model trained on solutions from MAENS, DARS shows the best approximation to MAENS regarding the values of MPE. For DARS, the Ptr-Net is capable to learn the mapping from CARP instances to the solutions, as shown by the good performance of DARS1 in Table 2. However, it may not find feasible solutions when the number of tasks increases, which is reported in [15]. The masking scheme is proposed to address this drawback and performs well.

The pre-sorting and post-sorting can improve the MPE which means better solutions. For pre-sorting it can be verified by comparing the values between DARS2 and DARS4, DARS3 and DARS. For post-sorting, the performance between DARS2 and DARS3, DARS4 and DARS in Table 2 implies its effectiveness.

Therefore, some conclusions can be derived based on the above discussions. The masking scheme can be effectively on filtering infeasible solutions, the Ptr-Net is capable to learn the mapping between CARP instances and their corresponding solutions; the post-sorting and the pre-sorting can both improve the solutions. These conclusions are coincident with the motivations described in Section III. Additionally, the results also imply the generalization ability of DARS as it performs well on the test sets where the CARP instances are unseen during the training.

2) SUPERVISED AND REINFORCEMENT LEARNING

According to Table 2, except DARS1 which may fail to find feasible solutions, all the other variants of DARS show better solution quality when compared with VRP-RL. This is intuitive, as heuristic methods are the state-of-the-art inexact methods. Thus, DARS that makes use of the experiences from the heuristic methods (i.e., the solutions generated by MAENS in the training set) are more likely to perform better than VRP-RL by learning from scratch. Nevertheless, reinforcement learning has the potential to perform better than heuristic methods, once it can explore the solution space more effectively considering the characteristics of combinatorial optimization tasks.

3) SPEED COMPARISON

The average computational time is shown in Table 3, and it is clear to see the advantages of neural solvers compared with MAENS, especially on GPU platforms. On CPUs, VRP-RL is close to DARS in terms of running time but worse in terms of solution quality considering Table 2. On GPUs, both VRP-RL and DARS show significant speedup. Due to the difficulty in the implementation of pre-sorting and post-sorting on GPUs, it is believed that the performance of DARS on GPU can be
TABLE 1. Information about datasets. $|E|$ is the number of edges, $T$ is the number of tasks, $Q$ is the capacity of each vehicle, demand is for each task edge. Value '$a - b$' indicates the column value for each CARP instance is randomly determined within the range from $a$ to $b$.

| Name   | $|E|$  | $T$  | $Q$  | demand |
|--------|------|------|------|--------|
| BJ-30  | 20-30| 5-29 | 20-25| 1-20   |
| BJ-50  | 30-50| 6-41 | 20-25| 1-20   |
| BJ-100 | 50-100| 13-72| 20-25| 1-20   |
| BJ-150 | 100-150| 28-100| 20-25| 1-20   |

TABLE 2. Comparison of -MPE (%) for different datasets and different neural solvers, the low the better. ‘√’ denotes the components involved in methods otherwise ‘−’. ‘Net’ denotes Ptr-Net, ‘Pre’ denotes pre-sorting, ‘MS’ denotes masking scheme, and ‘Post’ denotes post-sorting. Bold values are the best result for each dataset. ‘\’ means no feasible solutions are found.

| Method | Net | Pre | MS  | Post | BJ-30 | BJ-50 | BJ-100 | BJ-150 |
|--------|-----|-----|-----|------|-------|-------|--------|--------|
| DARS1  | √   | −   | −   | −    | 10.38 ± 10.12 | 14.37 ± 10.28 | 16.00 ± 5.97 | \     |
| DARS2  | √   | −   | √   | −    | 11.34 ± 10.35 | 15.51 ± 9.85  | 20.32 ± 9.37  | 16.20 ± 14.79 |
| DARS3  | √   | −   | √   | √    | 10.44 ± 9.84  | 14.13 ± 8.98  | 18.41 ± 8.40  | 15.02 ± 14.39 |
| DARS4  | √   | √   | √   | −    | 8.44 ± 9.01   | 10.20 ± 7.51  | 10.05 ± 5.66  | 13.58 ± 13.88 |
| DARS   | √   | √   | √   | √    | 7.92 ± 8.66   | 9.62 ± 7.12   | 9.50 ± 5.31   | 12.86 ± 13.66 |
| VRP-RL | −   | −   | −   | −    | 23.44 ± 16.12 | 25.30 ± 14.20 | 26.55 ± 11.93 | 16.53 ± 15.40 |

TABLE 3. Comparison of average computational time ($\times 10^{-4}$ seconds per instance). Values are denoted in forms of ‘$mean \pm std$’. MAENS shows extremely large stand deviation than neural solvers.

| Data   | MAENS CPU | VRP-RL CPU | DARS GPU |
|--------|-----------|------------|----------|
| BJ-30  | 1572 ± 1410 | 2549 ± 563  | 510 ± 195 |
| BJ-50  | 3943 ± 3723 | 4064 ± 860  | 901 ± 189 |
| BJ-100 | 13858 ± 16702 | 7283 ± 1776 | 897 ± 270 |
| BJ-150 | 33478 ± 59272 | 12924 ± 2306 | 1409 ± 252 |

further improved. This could be a direction for future work of faster sorting methods.

It should be note that, the stand deviations of MAENS are extremely large in Table 3. This is because MAENS conducts a randomized iterative search which may cost various time on different instances, while neural solvers direct map the instances to solutions and thus could be stable in the performance of time. This indicates that it is not easy to predict the time cost for heuristic methods for certain instances while neural solvers could. Specifically, we show the speedups of DARS over MAENS in Fig. 3. The speedup is calculated by the running time of MAENS divided by the running time of DARS for each instance and averaged over the test set. It should be noted that the standard deviations of the speedups are large due the unstable time performance of MAENS as discussed above.

The largest speedup for each dataset (i.e., the extreme cases) on GPUs are: 378 (BJ-30), 131 (BJ-50), 194 (BJ-100) and 357 (BJ-150). According to Table 3 and Fig. 3, DARS achieves the best approximation to MAENS.
and is significantly faster on GPUs. Note that some parts of DARS are implemented in python which is much slower than C++ used by MAENS.

4) COMPARED WITH MAENS WITH DIFFERENT SETTINGS
The running time of MAENS are mainly controlled by two hyper-parameters (ubstrail, max iteration) where each of them takes responsible for an independent while loop. The setting of (ubstrail, max iteration) used in previous experiments is (20, 50). Different MAENS with different running time are ran by varying the settings (ubstrail, max iteration). Another setting is examined in this experiment and the results are shown in Table 4, together with the performance of DARS. This experiment is run on BJ-100 dataset.

As shown in Table 4, the solution quality of DARS is better than MAENS with unsuitable configurations. As hyper-parameter tuning for heuristic algorithms is challenging, results in Table 4 indicates DARS can achieve a good tradeoff in terms of speed and the solution quality.

V. CONCLUSION AND FUTURE WORK
This paper proposes a novel approach called DARS to make CARP solving faster in a deep learning style. By utilizing solutions generated by the heuristic method as supervised information to train a DNN, the proposed approach has three benefits. Firstly, it makes the CARP solving as a direct functional mapping instead of an iterative searching process. Secondly, the supervised training can effectively utilize the experiences of heuristic methods that have developed for decades. Thirdly, the new CARP solver can be easily accelerated, since the acceleration of DNN has achieved rapid development in recent years.

Compared with the heuristic algorithm, the proposed method approximates the solutions given by the heuristic algorithm with a good performance and has a great advantage in speed, especially with the help of specific hardware such as GPUs. In extreme cases, DARS has achieved up to 357 times faster than the compared heuristic method. Generally, DARS has presented competitive results. Nevertheless, DARS also presents some limitations. One is that using the experience of existing heuristics can achieve speedup in problem solving, but it also limits the performance of the solution to some extent. Incorporating methods to automatically update the experience may help to further improve the performance. The other is that the four steps adopted by the algorithm facilitate the flexibility to combine existing state-of-the-art methods, but also leave room for improvement by integrating them synthetically and comprehensively. Besides, in order to further improve its performance, two directions are worthy of future study. One is to incorporate more advanced learning [37]–[39] and optimization mechanisms [40]–[42] into DARS and the other is to investigate GPU implementations [43] to further improve its efficiency.

TABLE 4. DARS with MAENS of different settings. ‘−’ indicates the value is not needed. ‘ub’ denotes ubstrail, and ‘max’ denotes max iteration. The results are averaged over 10 runs and dataset is BJ-100.

| Method     | ub | max | Time (s) | -MPE (%) |
|------------|----|-----|----------|----------|
| MAENS      | 20 | 50  | 1.399    | −        |
| MAENSv     | 1  | 1   | 0.028    | 12.480   |
| DARS CPU   | −  | −   | 0.099    | 9.500    |
| DARS GPU   | −  | −   | 0.081    | 9.500    |

FIGURE 3. Speedup of DARS over MAENS.

REFERENCES
[1] B. L. Golden and R. T. Wong, “Capacitated arc routing problems,” Networks, vol. 11, no. 3, pp. 305–315, 1981.
[2] S. Wohlk, “A decade of capacitated arc routing,” in The Vehicle Routing Problem: Latest Advances and New Challenges. Boston, MA, USA: Springer, 2008, pp. 29–48.
[3] W. Padungwese, J. Thompson, and R. Lewis, “Effects of update frequencies in a dynamic capacitated arc routing problem,” Networks, vol. 76, no. 4, pp. 522–538, Dec. 2020.
[4] E. Babaei Tirkolaee, I. Mahdavi, M. S. Seyyed Esfahani, and G.-W. Weber, “A hybrid augmented ant colony optimization for the multi-trip capacitated arc routing problem under fuzzy demands for urban solid waste management,” Waste Manage. Res., vol. 38, no. 2, pp. 156–172, Feb. 2020.
[5] P. Keenan, J. Panadero, A. A. Juan, R. Martí, and S. Mcgarraghy, “A strategic oscillation simheuristic for the time capacitated arc routing problem with stochastic demands,” Comput. Oper. Res., vol. 133, Sep. 2021, Art. no. 105377.
[6] R. Kendy Arakaki and F. Luiz Uberti, “An efficiency-based path-scanning heuristic for the capacitated arc routing problem,” Comput. Oper. Res., vol. 103, pp. 285–295, Mar. 2019.
[7] K. Tang, Y. Mei, and X. Yao, “Memetic algorithm with extended neighborhood search for capacitated arc routing problems,” IEEE Trans. Evol. Comput., vol. 13, no. 5, pp. 1151–1166, Oct. 2009.
[8] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, pp. 436–444, Feb. 2015.
[9] D. J. Miller, Z. Xiang, and G. Kesidis, “Adversarial learning targeting deep neural networks,” Proc. IEEE, vol. 108, no. 3, pp. 402–433, Mar. 2020.
[10] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Müller, “Evaluating the visualization of what a deep neural network has learned,” IEEE Trans. Neural Netw. Learn. Syst., vol. 28, no. 11, pp. 2660–2673, Nov. 2017.
[11] M. U. Younus, M. K. Khan, M. R. Anjum, S. Afridi, Z. A. Arain, and A. A. Jamali, “Optimizing the lifetime of software defined wireless sensor network via reinforcement learning,” IEEE Access, vol. 9, pp. 259–272, 2021.
VOLUME 10, 2022
4785

[12] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, “PuDianNao: A polyvalent machine learning accelerator,” ACM SIGPLAN Notices, vol. 50, no. 4, pp. 369–381, May 2015.

[13] E. Wang, J. J. Davis, R. Zhao, H. Ng, X. Niu, W. Luk, P. Y. K. Cheung, and G. A. Constantinides, “Deep neural network approximation for custom hardware: Where we’ve been, where we’re going,” ACM Comput. Surv., vol. 52, no. 2, p. 40, 2019.

[14] A. Grover and J. Leskovec, “Node2vec: Scalable feature learning for networks,” in Proc. Int. Conf. Knowl. Discovery Data Mining, B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, Eds., San Francisco, CA, USA, Aug. 2016, pp. 855–864.

[15] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 28, Montreal, QC, Canada, Dec. 2015, pp. 2692–2700.

[16] Y. Chen, J.-K. Hao, and F. Glover, “A hybrid Metaheuristic approach for the capacitated arc routing problem,” Eur. J. Oper. Res., vol. 253, no. 1, pp. 25–39, Aug. 2016.

[17] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in Proc. Int. Conf. Learn. Represent., San Diego, CA, USA, May 2015, pp. 1–15.

[18] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural combinatorial optimization with reinforcement learning,” in Proc. Int. Conf. Learn. Represent., Toulon, France, Apr. 2017, pp. 1–15.

[19] M. Nazari, A. Oroojlooy, L. V. Snyder, and M. Takić, “Reinforcement learning for solving the vehicle routing problem,” in Proc. Adv. Neural Inf. Process. Syst., Montréal, QC, Canada, Dec. 2018, pp. 9861–9871.

[20] W. Kool, H. van Hoof, and M. Welling, “Attention, learn to solve routing problems,” in Proc. 7th Int. Conf. Learn. Represent., New York, NY, USA, May 2019, pp. 1–25.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, Long Beach, CA, USA, Dec. 2017, pp. 5998–6008.

[22] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Mach. Learn., vol. 8, nos. 3–4, pp. 229–256, 1992.

[23] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, “Learning combinatorial optimization algorithms over graphs,” in Proc. Adv. Neural Inf. Process. Syst., Long Beach, CA, USA, Dec. 2017, pp. 6348–6358.

[24] H. Dai, B. Dai, and L. Song, “Discriminative embeddings of latent variable models for structured data,” in Proc. Int. Conf. Mach. Learn., vol. 48, M. Balcan and K. Q. Weinberger, Eds., New York, NY, USA, Jun. 2016, pp. 2702–2711.

[25] H. Li and G. Li, “Learning to solve capacitated arc routing problems by policy gradient,” in Proc. Congr. Evol. Comput. Wellington, New Zealand, Jun. 019, pp. 1291–1298.

[26] A. Milan, S. H. Rezatofighi, R. Garg, A. R. Dick, and I. D. Reid, “Data-driven approximations to NP-hard problems,” in Proc. Conf. Artif. Intell., S. P. Singh and S. Markovitch, Eds., San Francisco, CA, USA, Dec. 2017, pp. 1435–1449.

[27] M. O. R. Prates, P. H. C. Avelar, H. Lemos, L. C. Lamb, and M. Y. Vardi, “Learning to solve NP-complete problems: A graph neural network for decision TSP,” in Proc. Conf. Artif. Intell., Honolulu, HI, USA, Jan./Feb. 2019, pp. 4731–4738.

[28] W.-L. Pearn, A. Assad, and B. L. Golden, “Transforming arc routing into node routing problems,” Comput. Oper. Res., vol. 14, no. 4, pp. 285–288, Jan. 1987.

[29] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano, “A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems,” Prog. Artif. Intell., vol. 1, no. 1, pp. 103–117, 2012.

[30] E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numer. Math., vol. 1, no. 1, pp. 269–271, Dec. 1959.

[31] K. Tang, J. Wang, X. Li, and X. Yao, “A scalable approach to capacitated arc routing problems based on hierarchical decomposition,” IEEE Trans. Cybern., vol. 47, no. 11, pp. 3928–3940, Nov. 2017.

[32] H. Yousuf, M. Lahzi, S. A. Salloum, and K. Shaalan, “A systematic review on sequence-to-sequence learning with neural network and its models,” Int. J. Electr. Comput. Eng., vol. 11, no. 3, pp. 2315–2326, 2021.

[33] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, “Graph neural networks: A review of methods and applications,” AI Open, vol. 1, pp. 57–81, Dec. 2020.

[34] L. Feng, C. Zhao, and Y. Sun, “Dual attention-based encoder-decoder: A customized sequence-to-sequence learning for soft sensor development,” IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 8, pp. 3306–3317, Aug. 2021.

[35] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. 3rd Int. Conf. Learn. Represent., San Diego, CA, USA, May 2015, pp. 1–15.

[36] A. M. Saxe, J. L. McClelland, and S. Ganguli, “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks,” in Proc. Int. Conf. Learn. Represent., Banff, AB, Canada, Apr. 2014, pp. 1–22.

[37] W. Hong, P. Yang, Y. Wang, and K. Tang, “Multi-objective magnitude-based pruning for latency-aware deep neural network compression,” in Proc. Int. Conf. Parallel Problem Solving Nature, vol. 12269, Leiden, The Netherlands, Sep. 2020, pp. 470–483.

[38] P. Goyal and E. Ferrara, “Graph embedding techniques, applications, and performance: A survey,” Knowl.-Based Syst., vol. 151, pp. 78–94, Jul. 2018.

[39] M. G. Bellemare, W. Dabney, and R. Munos, “A distributional perspective on deep reinforcement learning,” in Proc. 34th Int. Conf. Mach. Learn., vol. 70, Sydney, NSW, Australia, Aug. 2017, pp. 449–458.

[40] W. Hong, K. Tang, A. Zhou, H. Ishibuchi, and X. Yao, “A scalable indicator-based evolutionary algorithm for large-scale multiobjective optimization,” IEEE Trans. Evol. Comput., vol. 23, no. 3, pp. 525–537, Jun. 2019.

[41] F. R. Al-Osaimi, “A sparse branch and bound optimization of noisy weighted DAG modification under constraints: A method for monocular data association to multiple laser planes,” IEEE Access, vol. 9, pp. 5994–6007, 2021.

[42] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, “Multiobjective evolutionary algorithms: A survey of the state of the art,” Swarm Evol. Comput., vol. 1, no. 1, pp. 32–49, 2011.

[43] M. A. Alomirat, Y. Jararweh, M. Al-Ayyoub, M. A. Shehab, and B. B. Gupta, “Accelerating compute intensive medical imaging segmentation algorithms using hybrid CPU-GPU implementations,” Multimedia Tools Appl., vol. 76, no. 3, pp. 3537–3555, Feb. 2017.

WENJING HONG (Member, IEEE) received the B.Eng. and Ph.D. degrees in computer science and technology from the University of Science and Technology of China, China, in 2012 and 2018, respectively. From 2019 to 2020, she was a Postdoctoral Researcher with the School of Management, University of Science and Technology of China. Since 2021, she has been a Research Assistant Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, China. Her research interests include evolutionary learning and multi-objective optimization.

TONGLIN LIU received the Ph.D. degree from the Shenyang Institute of Automation, Chinese Academy of Science, Shenyang, China, in 2010. From 2012 to 2014, he carried out Postdoctoral Research with the Institute of Aeronautical Technology, China Aerospace Science and Industry Corporation, Beijing, China. He is currently a Senior Engineer with the Beijing Electro-Mechanical Engineering Institute, Beijing. His current research interests include complex system design, modeling, assessment, and optimization.