A Learning-Based Framework for Constrained Shortest Path Problems

Xuefeng Jin[1] and Shunzheng Yu[1,*]

Abstract— **The doubly resource constrained elementary shortest path problem (DRCESPP) has important applications in intelligent network scenarios. For solving this strong NP-hard problem, we introduce learning-based techniques and present a solution framework integrating preprocessing, graph neural networks (GNNs) and deep reinforcement learning (DRL). First, the preprocessing procedure reduces the network size and provides initial feasible paths. Then, the classifier, implemented by a graph attention network (GAT), filters out the reduced networks where better paths exist after preprocessing. Finally, a DRL-based heuristic attempts to construct the optimal path for the filtered reduced networks with an end-to-end solution paradigm. We devise a crafted reward function and a shared low-variance baseline for the reinforcement learning optimization algorithm. Our experiments suggest that the proposed framework achieves better performance compared with competitive heuristic algorithms in terms of solution quality and computational efficiency.**

I. INTRODUCTION

The resource constrained shortest path problem (RCSPP) is a class of combinatorial optimization (CO) problems with practical applications [1], which consists of finding a minimum cost path between two predetermined nodes, subject to the constraint that the resource consumption along the path cannot exceed upper limits. The doubly resource constrained elementary shortest path problem (DRCESPP), where each resource consumption is constrained by a double sided inequality, is a general version of the RCSPP and is generally more difficult than the RCSPP. It has real-world applications in pricing and billing for differentiated Quality of Service (QoS) guarantees and customer's choice of cloud pools in intelligent scenarios such as computing power network, remote autonomous driving and Cloud VR.

Due to the NP-hard nature, the solution time of the exact algorithms is often unacceptable for the complex RCSPP problems, especially for the more difficult DRCESPPs (the cases with multiple double sided constraints). Therefore, heuristic algorithms are promising to find high-quality feasible solutions in a reasonable time, and achieve a trade-off between computational efficiency and solution quality. Some heuristic algorithms mainly including nature-inspired metaheuristics for solving RCSPPs can be found in the literature [2]–[8]. It is noteworthy that the work in [4] compares a number of classical metaheuristic algorithms, such as genetic algorithm (GA) [9], differential evolution (DE) [10], simulated annealing (SA) [11], ant colony optimization (ACO) [12], etc. Nevertheless, it has been recognized that these typical *iterative improvement heuristics* implicitly assume that feasible solutions to a given problem can be easily obtained and are not suitable for the CO problems with rare feasible solutions [13]. The DRCESPP we focus on is such a problem with an extremely small feasible space, especially for the case with multiple resource constraints and tight resource limits. Furthermore, the iteration-based heuristics do not yield experience that can be learned and exploited by other instances, which results in a completely new and computationally expensive iterative solution for each problem instance. In light of this, it is necessary and significant to explore a new and effective solution paradigm.

In recent years, learning-based algorithms, especially deep reinforcement learning (DRL) algorithms, have drawn considerable attention and made great progress in solving various CO problems [14]–[17]. It is generally accepted that heuristics can be learned end-to-end and parameterized using deep neural networks (DNNs) [18]. Such DNN model can infer good results for unseen problem instances in a short time once it has been well trained offline.

In this work, we aim at introducing learning-based techniques to solve DRCESPPs, and design a solution framework integrating preprocessing, graph neural networks (GNNs) and DRL. In the framework, the problem instance is first fed into a preprocessing procedure and is reduced in size, and then a classifier model implemented by a graph attention network (GAT) determines whether a better solution exists in the reduced network. If so, a decision model (a DRL-based *constructive heuristic*) attempts to find the potential optimal solution by sequentially constructing a path without numerous iterations. To make the DRL model training stable and fast, we devise a shared low-variance baseline which is derived from the paths sampled in both forward and backward directions. Our main contributions can be summarized as follows.

- Our work provides a novel and effective framework of solving DRCESPPs. To our best knowledge, we are the first to introduce learning-based techniques combining DRL and GNNs to solve the complex shortest path problem constrained with multiple double sided inequalities.
- For the DRL-based decision model, we devise crafted reward functions to encourage the agent to explore search space and satisfy multiple constraints, and present a shared baseline to stabilize training and accelerate convergence.
- The proposed framework has significant advantages over the iteration-based heuristics in terms of solution quality and computational efficiency, which reaffirms

[1]Xuefeng Jin and Shunzheng Yu are with School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. (* corresponding author, email: `jinxf@mails.ccnu.edu.cn`, `syu@mail.sysu.edu.cn`)

the view that traditional iteration-based heuristics are not suitable for solving problems with rare feasible solutions.

II. Preliminaries

A. Problem Formulation

Consider a directed cyclic graph $G = (V, E)$, where V and E are the set of nodes and directed arcs (edges), respectively. K nonnegative features are associated with each arc $(i, j) \in E$, the first of which is the arc cost, and the rest are $K-1$ different resource consumptions along the arc. Let w_{ij}^k represent the k-th scalar feature of arc (i, j). A path $\pi = \pi_{0:n} = (\pi_0 = v_s, \pi_1, ..., \pi_n = v_t)$ of length n from node v_s to v_t is a sequence of nodes such that $(\pi_{i-1}, \pi_i) \in E$ holds for all $i \in \{1, ..., n\}$. Let $W^k(\pi) = \sum_{(i,j)\in E(\pi)} w_{ij}^k$ be the total quantity of the k-th weight feature (cost or resource) accumulated along the path π, where $E(\pi)$ denotes the set of all arcs on π.

The doubly resource constrained elementary shortest path problem (DRCESPP) consists of finding a least cost elementary path (no node is repeated) from a source node v_s to a destination node v_t such that the cumulative quantity of each resource consumed along the path belongs to a specific interval (a double sided inequality), which can be briefly expressed as: minimize $W^1(\pi)$ such that $L_k \leq W^k(\pi) \leq U_k$ for all $k = 2, ..., K$, where L_k and U_k represent the lower and upper limits on the $(k-1)$-st resource consumption.

B. Preprocessing Techniques

Preprocessing is a powerful technique, which can not only significantly reduce network size (i.e., eliminate unpromising nodes and edges that cannot possibly lie on any feasible or optimal path), but also generate feasible solutions and even solve the problem directly to optimality. The main idea is to calculate the maximum or minimum cumulative weight feature of all paths passing through a node or arc based on the forward and reverse shortest path trees or some estimation methods, and then check whether the corresponding limits (e.g., resource limits and cost bounds) are reached. The reader is referred to [19] for basic preprocessing techniques. Our comprehensive preprocessing techniques can be found at https://github.com/hellomuyi/DRCESPP.

C. Pruning Strategies

Pruning strategies used in path construction are capable of discarding unpromising search space in Pulse algorithm [20] and increasing the success rate of constructing feasible paths in our decision model. In the process of path construction, a partial path $\pi_{0:t-1} = (\pi_0 = v_s, \pi_1, \pi_2, \ldots, \pi_{t-1})$ is being extended to its successor node π_t. Obviously, $\pi_t \in \mathcal{N}_{\pi_{t-1}} \backslash V(\pi_{0:t-1})$ because of the elementarity constraints, where $\mathcal{N}_{\pi_{t-1}} = \{v_j \in V | (\pi_{t-1}, j) \in E\}$ denotes the set of the first-order outgoing neighbors of node π_{t-1}, and $V(\pi_{0:t-1})$ is the set of all nodes on $\pi_{0:t-1}$. With the help of two additional pruning strategies, namely infeasibility pruning and optimality pruning (for details, see the Pulse algorithm [20]), we can further shrink the set of successor nodes and discard successor node v_j from $\mathcal{N}_{\pi_{t-1}} \backslash V(\pi)$ if $W^k(\pi_{0:t-1}) + w_{\pi_{t-1}j}^k + W^k(\overleftarrow{\pi}_j^k) > U_k$ for some $k \in \{2, 3, \ldots, K\}$ or $W^1(\pi_{0:t-1}) + w_{\pi_{t-1}j}^1 + W^1(\overleftarrow{\pi}_j^1) \geq UB$, where $\overleftarrow{\pi}_j^k$ represents the shortest path from v_j to the destination node v_t in the k-th feature network, and UB denotes cost of the best-so-far path. Let $\hat{\mathcal{N}}_{\pi_{0:t-1}} \subseteq \mathcal{N}_{\pi_{t-1}}$ denote the set of unpruned successor nodes of $\pi_{0:t-1}$ after the pruning strategies.

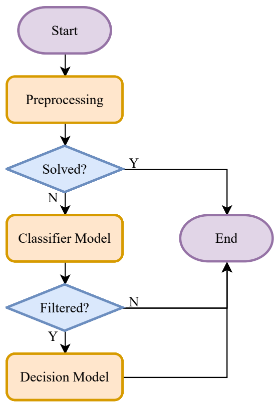

Fig. 1. The overall framework flowchart.

III. Proposed Method

Our proposed learning-based framework consists of a preprocessing procedure, a classifier model and a decision model, as illustrated in Fig. 1. The preprocessing procedure takes the problem instance as input and outputs the reduced network with fewer nodes and edges. Also, some feasible paths may be generated during the preprocessing, the best of which (its cost value is denoted as UB) will be used as the threshold for classification in the classifier model and as the cost upper limit for optimality pruning in the decision model. Note that we can conclude that the problem has been solved to optimality as soon as any termination test takes effect (e.g., all edges or intermediate nodes are eliminated) in the preprocessing. For the unsolved reduced networks after preprocessing, the classifier model will filter out those where a better path (its cost value is less than UB) exists, and then the decision model attempts to find the potential optimal solution by sequentially constructing a path. As for those unfiltered networks, they are judged to have been solved with UB as the optimal cost value.

Both the classifier model and decision model have an encoder-decoder structure. The encoder maps input features to higher-level representations. The decoder generates a prediction or a decision based on these representations and task-specific context information. Fig. 2 gives the overview of our encoder-decoder model architecture.

A. Input Features

Let $X = \{X_i | v_i \in V\}$ be node feature matrix, where X_i denotes the feature vector of node v_i. Analogously, let $E = \{E_{ij} | (i, j) \in E\}$ be edge feature matrix, where E_{ij} is the feature vector of edge (i, j). We concatenate the information about weight features of adjacency edges and the

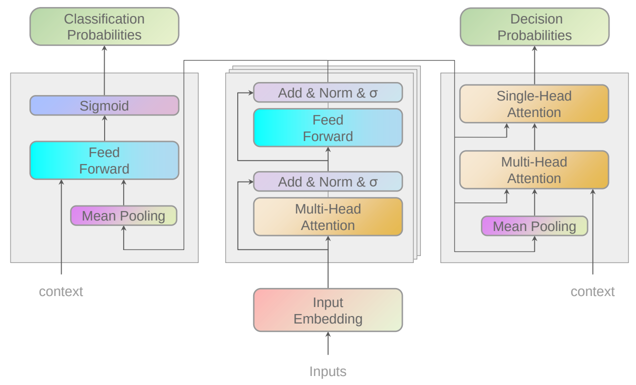

Fig. 2. Overview of encoder-decoder architecture. The classifier model (left) and the decision model (right) use the same encoder structure (middle), but with their own parameters.

(reverse) shortest path trees on each feature subgraph as the primary node features. Specifically, the $(5 \cdot K)$-dimensional node feature X_i is obtained by:

$$X_i = \big\Vert_{k=1}^{K} \Big(\frac{\sum_{j \in \mathcal{N}_i} w_{ij}^k}{|\mathcal{N}_i|} \,\Vert\, W^k(\overrightarrow{\pi}_i^k) \,\Vert\, |\overrightarrow{\pi}_i^k| \,\Vert\, W^k(\overleftarrow{\pi}_i^k) \,\Vert\, |\overleftarrow{\pi}_i^k| \Big) \tag{1}$$

where $\Vert$ denotes the concatenation operation, $\mathcal{N}_i$ is the set of the first-order outgoing neighbors of v_i, and $\overrightarrow{\pi}_i^k$ and $\overleftarrow{\pi}_i^k$ represent the shortest path from the source node v_s to the node v_i and the shortest path from v_i to the destination node v_t in the k-th feature network, respectively. The primary edge features E_{ij} are a K-dimensional vector of weights w_{ij}^k:

$$E_{ij} = \big\Vert_{k=1}^{K} w_{ij}^k \tag{2}$$

All raw features X and E are scaled without a shift before being fed into the model.

B. Encoder: Graph Representation Learning

Graph representation learning, also called graph embedding, aims at encoding nodes (or edges or entire (sub)graphs) into a low-dimensional vector space, which is the basis for downstream machine learning tasks. Based on the strong ability of GNNs to extract latent representations from graph data lying on non-Euclidean spaces, we use GNNs as the representation learning module, also known as an *encoder*.

Specifically, our encoder consists of multiple stacked identical layers, and each layer has a multi-head attention (MHA) sublayer and a fully connected feed-forward (FF) sublayer, which are adopted from [18]. Also, each sublayer adds a residual connection (a skip connection) [21], batch normalization (BN) [22] and nonlinear activation σ. The middle part of Fig. 2 gives an overview of the encoder structure.

Let $h_i^{(\ell)}$ and $h_{ij}^{(\ell)}$ denote the representation outputs of layer ℓ for node v_i and edge (i,j), respectively, where $\ell \in \{1, 2, ..., L\}$ and L is the number of GNN layers. Besides, $h_i^{(0)} = W_v^{(0)} X_i + b_v$ and $h_{ij}^{(0)} = W_e^{(0)} E_{ij} + b_e$ are initial d_v-dimensional and d_e-dimensional embeddings obtained via an affine transformation with parameters $W_v^{(0)} \in \mathbb{R}^{d_v \times 5K}$, $W_e^{(0)} \in \mathbb{R}^{d_e \times K}$, $b_v \in \mathbb{R}^{d_v}$ and $b_e \in \mathbb{R}^{d_e}$, where the parameters with subscript v (resp. e) indicate that they are shared for every node (resp. edge). After all the L layers, the encoder outputs the final node representations $h_i^{(L)}$ and edge representations $h_{ij}^{(L)}$. The graph embedding, denoted as $\bar{h}^{(L)}$, is represented by averaging all the individual node and edge representations at the last layer:

$$\bar{h}^{(L)} = \frac{1}{|V|} \sum_{v_i \in V} h_i^{(L)} \,\Big\Vert\, \frac{1}{|E|} \sum_{(i,j) \in E} h_{ij}^{(L)} \tag{3}$$

Briefly, the encoder layer ℓ learns higher-level representations $h^{(\ell)}$ from $h^{(\ell-1)}$ as follows:

$$\hat{h}_i^{(\ell)} = \sigma\Big(\mathrm{BN}_v^{(\ell)}(h_i^{(\ell-1)} + \mathrm{MHA}_i^{(\ell)}(h_v^{(\ell-1)}, h_e^{(\ell-1)}))\Big) \tag{4}$$

$$h_i^{(\ell)} = \sigma\Big(\mathrm{BN}_v^{(\ell)}(\hat{h}_i^{(\ell)} + \mathrm{FF}_v^{(\ell)}(\hat{h}_i^{(\ell)}))\Big) \tag{5}$$

$$\hat{h}_{ij}^{(\ell)} = \sigma\Big(\mathrm{BN}_e^{(\ell)}(h_{ij}^{(\ell-1)} + \mathrm{MHA}_{ij}^{(\ell)}(h_v^{(\ell-1)}, h_e^{(\ell-1)}))\Big) \tag{6}$$

$$h_{ij}^{(\ell)} = \sigma\Big(\mathrm{BN}_e^{(\ell)}(\hat{h}_{ij}^{(\ell)} + \mathrm{FF}_e^{(\ell)}(\hat{h}_{ij}^{(\ell)}))\Big) \tag{7}$$

Next, by dissecting the MHA sublayer and the FF sublayer, we elaborate on how to learn higher-level node and edge representations in detail.

1) Multi-Head Attention (MHA) Sublayer: First, an edge-integrated attention mechanism $a : \mathbb{R}^{d_v} \times \mathbb{R}^{d_v} \times \mathbb{R}^{d_e} \to \mathbb{R}$ is used to compute attention coefficients $u_{ij}^{(\ell)}$ between node v_i and its neighbor v_j at layer ℓ:

$$u_{ij}^{(\ell)} = a^{(\ell)}\Big(h_i^{(\ell-1)}, h_j^{(\ell-1)}, h_{ij}^{(\ell-1)}\Big) \tag{8}$$

where $W_v^{(\ell)} \in \mathbb{R}^{d_v \times d_v}$ and $W_e^{(\ell)} \in \mathbb{R}^{d_e \times d_e}$ are shared weight matrices for every node and edge, respectively. Then, the attention coefficients are normalized by the softmax function:

$$\alpha_{ij}^{(\ell)} = \mathrm{softmax}_j(u_{ij}^{(\ell)}) = \frac{\exp(u_{ij}^{(\ell)})}{\sum_{k \in \mathcal{N}_i} \exp(u_{ik}^{(\ell)})} \tag{9}$$

where α_{ij} are (normalized) attention weights, $\mathcal{N}_i$ is the set of the first-order outgoing neighbors of v_i (including itself). Through the one-hop neighborhood, the graph structure information is injected into the attention mechanism.

After some testing, we find that the *scaled dot-product attention* proposed in [23] is significantly better than the *additive attention* in the GAT [24] and EGAT [25] or the hybrid attention using both mechanisms. The attention weights under the scaled dot-product attention can be reformulated as follows:

$$\alpha_{ij}^{(\ell)} = \frac{\exp\Big(\big(W_Q^{(\ell)} h_i^{(\ell-1)}\big)^T \big(W_K^{(\ell)}(h_j^{(\ell-1)} \Vert h_{ij}^{(\ell-1)})\big) / \sqrt{d_v}\Big)}{\sum_{k \in \mathcal{N}_i} \exp\Big(\big(W_Q^{(\ell)} h_i^{(\ell-1)}\big)^T \big(W_K^{(\ell)}(h_k^{(\ell-1)} \Vert h_{ik}^{(\ell-1)})\big) / \sqrt{d_v}\Big)} \tag{10}$$

where $W_Q^{(\ell)} \in \mathbb{R}^{d_v \times d_v}$ and $W_K^{(\ell)} \in \mathbb{R}^{d_v \times (d_v + d_e)}$ are learnable parameter matrices, $\cdot^T$ denotes the transposition operation and $\Vert$ denotes the concatenation operation. Using attention based weighted summation, the output of the attention sublayer w.r.t. node v_i is computed by aggregating the representations of its neighbourhood (including adjacency nodes and edges):

$$h_{MHA_i}^{(\ell)} = \sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(\ell)} \Big(W_V^{(\ell)} \big(h_j^{(\ell-1)} \parallel h_{ij}^{(\ell-1)} \big) \Big) \tag{11}$$

Here, $W_V^{(\ell)} \in \mathbb{R}^{d_v \times (d_v + d_e)}$ is set to guarantee the same node representation dimension between sublayers.

As for edge representation learning, a popular method is to exploit the line graph [26]–[28], where edges are treated as nodes and one can update edge representations just like updating node representations in the original graph. However, the scale of the line graph is often much larger than the original graph, resulting in a huge computational burden. Therefore, we adopt the node-transit strategy proposed in [25] to update the edge representations using a multilayer perceptron (MLP), based on a quintet about the edge:

$$\beta_{ij}^{(\ell)} = \mathrm{softmax}_j \Big(\big(W_Q'^{(\ell)} h_i^{(\ell-1)} \big)^T \big(W_K'^{(\ell)} (h_j^{(\ell-1)} \parallel h_{ij}^{(\ell-1)}) \big) / \sqrt{d_v} \Big) \tag{12}$$

$$h_{ei}^{(\ell)} = \sum_{j \in \mathcal{N}_i} \beta_{ij}^{(\ell)} W_V'^{(\ell)} \big(h_j^{(\ell-1)} \parallel h_{ij}^{(\ell-1)} \big) \tag{13}$$

$$h_{MHA_{ij}}^{(\ell)} = \mathrm{MLP}\big(h_i^{(\ell-1)}, h_j^{(\ell-1)}, h_{ij}^{(\ell-1)}, h_{ei}^{(\ell)}, h_{ej}^{(\ell)} \big) \tag{14}$$

where $h_{e_i}^{(\ell)}$ in (13) represents the attention based weighted summation of adjacency node representations and edge representations of v_i at $(\ell-1)$-st layer. In the experiments, MLP has one layer with a weight matrix $W \in \mathbb{R}^{d_e \times (2d_v + 3d_e)}$ and a bias vector $b \in \mathbb{R}^{d_e}$.

In our experiments, the attention mechanism involved in the encoder has M independent heads, and outputs a concatenation of the representations learned by each head. For instance, (11) is reformulated under multiple heads as follows:

$$h_{MHA_i}^{(\ell)} = \Big\Vert_{m=1}^{M} \sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(\ell,m)} \Big(W_V^{(\ell,m)} \big(h_j^{(\ell-1)} \parallel h_{ij}^{(\ell-1)} \big) \Big) \tag{15}$$

where $W_V^{(\ell,m)} \in \mathbb{R}^{\frac{d_v}{M} \times (d_v + d_e)}$.

2) Feed-Forward (FF) Sublayer: The FF sublayer with one hidden layer with dimension d_{ff} and a ReLU activation takes as input node representations $\hat{h}_i^{(\ell)}$ derived from (4) or edge representations $\hat{h}_{ij}^{(\ell)}$ derived from (6), and outputs:

$$h_{FF_i}^{(\ell)} = W_v^{(\ell,ff_1)} \cdot \mathrm{ReLU}\big(W_v^{(\ell,ff_0)} \hat{h}_i^{(\ell)} + b_v^{(\ell,ff_0)} \big) + b_v^{(\ell,ff_1)} \tag{16}$$

$$h_{FF_{ij}}^{(\ell)} = W_e^{(\ell,ff_1)} \cdot \mathrm{ReLU}\big(W_e^{(\ell,ff_0)} \hat{h}_{ij}^{(\ell)} + b_e^{(\ell,ff_0)} \big) + b_e^{(\ell,ff_1)} \tag{17}$$

C. Classifier Model

The classifier model is embedded to filter the preprocessed networks that may yield better solutions. This is a typical graph-level binary classification task.

After encoding, we further inject task-specific context information (here, i.e., the $(4K)$-dimensional resource limits L_k and U_k and 1-dimensional current optimal cost UB) into the graph embedding $\bar{h}^{(L)}$, and thus obtain an auxiliary context node (denoted as node v_c) embedding $h_c^{(L)} \in \mathbb{R}^{d_v + d_e + 4K + 1}$. Finally, the classification probability is output via an FF layer similar to (16) followed by a sigmoid activation:

$$p = \mathrm{sigmoid}\big(FF(h_c^{(L)}) \big) \tag{18}$$

The classifier model is trained by supervised learning, based on the labels obtained by our improved pulse algorithm (cf. https://github.com/hellomuyi/DRCESPP) and binary cross entropy loss:

$$\mathcal{L} = -\frac{1}{B} \sum_{i=1}^{B} \left[y_i \cdot \log\left(p(s_i)\right) + (1 - y_i) \cdot \log\left(1 - p(s_i)\right) \right] \tag{19}$$

where B is the size of mini-batch, s_i is a problem instance.

D. Decision Model

1) Decoding Process: The decoder sequentially constructs a path $\pi = (\pi_0 = v_s, \pi_1, \pi_2, \ldots, \pi_n)$ of length n from the source node v_s by iterating at most $(|V| - 1)$ times (discrete time steps). At a certain time step $t \in \{1, 2, \ldots\}$, for the already constructed partial path $\pi_{0:t-1} = (\pi_0 = v_s, \pi_1, \ldots, \pi_{t-1})$ on instance s, the encoder-decoder decision model produces a probability distribution p over next-hop nodes π_t of $\pi_{0:t-1}$, which is defined as follows:

$$p = p_{\boldsymbol{\theta}}\big(\pi_t | s, \pi_{0:t-1}, \pi_t \in \hat{\mathcal{N}}_{\pi_{0:t-1}} \big) \tag{20}$$

where $\boldsymbol{\theta}$ is the learnable parameters of decision model, and $\hat{\mathcal{N}}_{\pi_{0:t-1}}$ is the set of successor nodes that are not pruned by the pruning techniques (pruning by elementarity, optimality and infeasibility on upper limits) discussed in section II-C. That is, we leverage the information gained from the preprocessing phase to discard the unpromising successors that cannot possibly lie on any optimal or resource upper limit feasible path, thus increasing the probability of constructing feasible or optimal paths.

Based on the probability distribution in (20), the decoder can construct a path sequentially from time step $t = 1$ by greedy decoding or sampling or other methods, with the joint probability as follows:

$$p_{\boldsymbol{\theta}}(\pi | s) = \prod_{t=1}^{n} p_{\boldsymbol{\theta}}(\pi_t | s, \pi_{0:t-1}, \pi_t \in \hat{\mathcal{N}}_{\pi_{0:t-1}}) \tag{21}$$

The sequential construction will terminate at time step t, once the specified target node v_t is reached (i.e., $\pi_t = v_t$) or there are no unpruned successors that can be extended (i.e., $\hat{\mathcal{N}}_{\pi_{0:t-1}} = \emptyset$). It is noteworthy that the constructed path may not be feasible, either failing to reach v_t due to $\hat{\mathcal{N}}_{\pi_{0:t-1}} = \emptyset$ and $\pi_{t-1} \neq v_t$, or reaching v_t but violating some resource lower limits. In the following, the calculation of the probability distribution of the decoding process is disassembled.

For the already constructed partial path $\pi_{0:t-1}$ at the time step t, the dynamic task-specific context information includes the graph embedding, the embedding of current node π_{t-1} (the last node of the partial path), the embedding of target node and the information about the resource consumptions of the partial path and the corresponding lower and upper limits.

Therefore, the auxiliary context node v_c is represented as a $(3d_v + d_e + 2(K-1))$-dimensional vector as follows.

$$h_c^{(L)} = \bar{h}^{(L)} \,\|\, h^{(L)}_{\pi_{t-1}} \,\|\, h^{(L)}_{v_t} \,\|\, \Big(\mathop{\Big\|}_{k=2}^{K} \big(L_k - W^k(\pi_{0:t-1})\big)\Big) \,\|\, \Big(\mathop{\Big\|}_{k=2}^{K} \big(U_k - W^k(\pi_{0:t-1})\big)\Big) \tag{22}$$

The higher-level context embedding $h_c^{(L+1)}$ is learned by a so-called glimpse operation, similar to the MHA sublayer:

$$\alpha_{cj} = \frac{\exp\Big(\big(W_Q^{(L+1)} h_c^{(L)}\big)^T \big(W_K^{(L+1)}(h_j^{(L)} \,\|\, h^{(L)}_{\pi_{t-1}j})\big)/\sqrt{d_v}\Big)}{\sum_{k\in\hat{\mathcal{N}}_{\pi_{0:t-1}}} \exp\Big(\big(W_Q^{(L+1)} h_c^{(L)}\big)^T \big(W_K^{(L+1)}(h_k^{(L)} \,\|\, h^{(L)}_{\pi_{t-1}k})\big)/\sqrt{d_v}\Big)} \tag{23}$$

$$h_c^{(L+1)} = \sum_{j\in\hat{\mathcal{N}}_{\pi_{0:t-1}}} \alpha_{cj} \Big(W_V^{(L+1)} \big(h_j^{(L)} \,\|\, h^{(L)}_{\pi_{t-1}j}\big)\Big) \tag{24}$$

where the superscript $(L+1)$ represents the first attention layer of decoder on top of the L-layer encoder. In practice, $h_c^{(L+1)} \in \mathbb{R}^{d_v}$ is still calculated in a multi-head attention way. Then, we use the second decoder layer (labeled with superscript $(L+2)$) with a single-head attention to output the probability distribution, and clip the attention coefficients into $[-C, C]$ $(C = 10)$ using a tanh activation function.

$$u_{cj} = C \cdot \tanh\Bigg(\frac{\big(W_Q^{(L+2)} h_c^{(L+1)}\big)^T \big(W_K^{(L+2)} (h_j^{(L)} \,\|\, h^{(L)}_{\pi_{t-1}j})\big)}{\sqrt{d_v}} \Bigg) \tag{25}$$

$$p_{\boldsymbol{\theta}}\big(\pi_t = v_j | s, \pi_{0:t-1}, \pi_t \in \hat{\mathcal{N}}_{\pi_{0:t-1}}\big) = \frac{\exp(u_{cj})}{\sum_{k\in\hat{\mathcal{N}}_{\pi_{0:t-1}}} \exp(u_{ck})} \tag{26}$$

2) Optimization Algorithm: For a given instance s, we define the reward function $r(\pi|s)$ of a path $\pi = (\pi_0 = v_s, \pi_1, \ldots, \pi_n)$ with length n as follows:

$$r(\pi|s) = r_{int}(\pi|s) + \begin{cases} -1/n & \text{if } \pi_n \neq v_t \\ \min\limits_{k\in\{2,3,\ldots,K\}} \{\frac{W^k(\pi)}{L_k} - 1\} \cdot 10/n & \text{if } \pi_n == v_t \\ \text{ and } W^k(\pi) < L_k \text{ for some } k & \\ 2 + 1 \cdot max\left\{\left(1 - \frac{W^1(\pi)}{UB}\right), -1\right\} & \text{otherwise} \end{cases} \tag{27}$$

where $r_{int}(\pi|s) = C(\pi|s)^{-1/2}$ is the intrinsic reward and $C(\pi|s)$ is the visit count of π during the training, $W^k(\pi)$ is the total quantity of the k-th weight feature (cost or resource) accumulated along π, L_k is the lower limit of resource k, and UB is the cost value of the best known solution generated from the preprocessing phase. The basic principle of designing reward function is to assign positive rewards to feasible paths and negative rewards to infeasible paths. In addition, the better the feasible path, the higher the reward. For infeasible paths, we attempt to increase the path length to explore more space and then find feasible paths via optimization, leading to the reward being proportional to path length. The count-based bonus r_{int} guides the decision model to explore the search space sufficiently.

The loss function (negative reward expectation) is defined by $\mathcal{L}(\boldsymbol{\theta}|s) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\pi|s)}\,[-r(\pi|s)]$. We use a model-free policy-based reinforcement algorithm, the REINFORCE algorithm [29] with baseline to optimize $\mathcal{L}$ and train the model by gradient descent.

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}|s) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\pi|s)} \left[-\big(r(\pi|s) - b(s)\big) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\pi|s) \right] \tag{28}$$

$b(s)$ is the baseline to reduce the gradient variance and stabilize training. For an instance s_i, based on the probability distribution of the decision model output, we sample N forward paths $\{\pi_i^1, \pi_i^2, \ldots \pi_i^N\}$ from v_s in the original graph G and N backward paths $\{\overleftarrow{\pi}_i^1, \overleftarrow{\pi}_i^2, \ldots \overleftarrow{\pi}_i^N\}$ from v_t in the reverse graph G' (obtained by reversing the direction of all arcs of G). Then, Our devised shared baseline is calculated by averaging the reward of all $2N$ paths:

$$b(s_i) = \frac{1}{2N} \sum_{j=1}^{N} \left[r(\pi_i^j|s_i) + r(\overleftarrow{\pi}_i^j|\overleftarrow{s_i}) \right] \tag{29}$$

The gradient can be approximated by Monte Carlo sampling as follows:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \approx \frac{1}{2BN} \sum_{i=1}^{B} \sum_{j=1}^{N} \Big[\big(b(s_i) - r(\pi_i^j|s_i)\big) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\pi_i^j|s_i) + \big(b(s_i) - r(\overleftarrow{\pi}_i^j|\overleftarrow{s_i})\big) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\overleftarrow{\pi}_i^j|\overleftarrow{s_i}) \Big] \tag{30}$$

Algorithm 1: REINFORCE with Shared Baseline

```
Input: number of epochs $E$, number of iterations $T$,
       batch size $B$, number of sampled paths in one
       direction $N$
Output: The decision model parameters $\boldsymbol{\theta}$
1  Initialize $\boldsymbol{\theta}$
2  for $epoch = 1, \ldots, E$ do
3      for $iteration = 1, \ldots, T$ do
4          $s_i \leftarrow \text{RandomInstance}(), \forall i \in \{1, \ldots, B\}$
5          $\pi_i^j \leftarrow \text{ForwardSamplePath}(s_i, p_{\boldsymbol{\theta}}), \forall j \in \{1, \ldots, N\}, \forall i \in \{1, \ldots, B\}$
6          $\overleftarrow{\pi}_i^j \leftarrow \text{BackwardSamplePath}(\overleftarrow{s_i}, p_{\boldsymbol{\theta}}), \forall j \in \{1, \ldots, N\}, \forall i \in \{1, \ldots, B\}$
7          $b(s_i) \leftarrow \frac{1}{2N} \sum_{j=1}^{N} [r(\pi_i^j|s_i) + r(\overleftarrow{\pi}_i^j|\overleftarrow{s_i})]$
8          $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \leftarrow \frac{1}{2BN} \sum_{i=1}^{B} \sum_{j=1}^{N} \Big[\big(b(s_i) - r(\pi_i^j|s_i)\big) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\pi_i^j|s_i) + \big(b(s_i) - r(\overleftarrow{\pi}_i^j|\overleftarrow{s_i})\big) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\overleftarrow{\pi}_i^j|\overleftarrow{s_i})\Big]$
9          $\boldsymbol{\theta} \leftarrow \text{Adam}\big(\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\big)$
10     end
11 end
```

where B is the size of mini-batch. The optimization goal is to minimize the loss (maximize the reward expectation) by assigning high probability to the better paths and low probability to the worse and infeasible paths.

The training algorithm is presented in Algorithm 1.

IV. Experiments

In this section, we present experiments on randomly generated instances with multiple double sided constraints to evaluate the performance of the proposed solution framework for DRCESPPs. Our framework was implemented in Python and the experiments were executed on a 64-bit Linux system configured with an 18-core Intel Core i9-7980XE CPU at 2.60 GHz and 128 GB of RAM. We implemented our classifier model and decision model using PyTorch and trained them through single NVIDIA GeForce RTX 2080 Ti GPU. The GATs in the encoder-decoder model architecture were implemented on the top of the components of PyTorch Geometric (PyG) [30], a GNN library that implements the Message Passing Neural Networks (MPNN) [31] framework by gather and scatter operations.

A. Datasets

We adopt the scale-free network based on the Barabási–Albert preferential attachment model [32] as the network topology, with size of $|V| = 100$ and $|E| = 4758$. The feature number K is set to 5, and the values w_{ij}^k are random numbers uniformly distributed in $[0, 1]$. Resource lower and upper limits L_k and U_k are set by $L_k = W^k(\overrightarrow{\pi}_t^1)+\epsilon$ and $U_k = 1.5W^k(\overrightarrow{\pi}_t^1)-0.5W^k(\overrightarrow{\pi}_t^k)-\epsilon$, where ϵ is a small positive number for clear interval boundaries.

For the classifier model, we randomly generate 12 800 instances and divided them into training set, validation set and test set in the ratio of 3:1:1. The number of positive and negative samples in each dataset is the same to ensure class balance. Three thousand two hundred instances where a better solution still exists after the preprocessing are used to train the decision model. We used 1000 unseen instances to evaluate the solution framework.

B. Hyperparameters

The embedding dimensions d_v and d_e of both nodes and edges are set to 128. We use $M = 8$ attention heads and $L = 1$ layer encoder for classifier model and $L = 3$ for decision model. The model parameters are initialized with the kaiming uniform (He initialization) [33]. We adopt the Adam optimizer with learning rate of 10^{-4} and clip the L_2 norm of gradients to 1. Dropout with drop rate of 0.1 is applied due to the potential overfitting. The number of sampled paths in one direction N is set to 5.

C. Evaluations

We first conducted separate experiments to evaluate the performance of the classifier model. The training was terminated according to the early stopping strategy on the validation set, and the trained model achieved a classification accuracy of 90.47% on the test set.

TABLE I
Summary of comparison results

Method	No. feasible			Average time (s)
ACS (α=0.01)	730			11.31
ACS (α=0.1)	749			6.86
ACS (α=0.5)	760			6.72
ACS (α=0.9)	759			6.66
DRW (α=0.01)	747			10.83
DRW (α=0.1)	744			8.14
DRW (α=0.5)	749			6.71
DRW (α=0.9)	741			6.46
	No. solved in preprocessing	classifier model	decision model	
PCD (N=128)	264	222/736	0.5631	0.63
PCD (N=256)			0.5946	1.12
PCD (N=512)			0.6081	2.10
PCD (N=1024)			0.6351	4.08

The proposed framework consisting of preprocessing, classifier model and decision model (henceforth, PCD) is compared with the following two representative algorithms. 1) Ant Colony System (ACS) [34], an improved version of Ant System (AS) [35], is one of the most effective and well-known nature-inspired algorithms in CO problems. 2) Deterministic Random Walk Model (henceforth, DRW) [2] searches the graph based on node fitness which is updated by the better paths found. Both algorithms require a maximum possible path cost τ_0 to calculate the initial pheromone level or node fitness. We set $\tau_0 = \alpha \cdot \text{EST}$, where $\text{EST} = min\{\sum_{v_i \in V\setminus\{v_t\}} \underset{(i,j)\in E}{max}\, w_{ij}^1, \sum_{v_j \in V\setminus\{v_s\}} \underset{(i,j)\in E}{max}\, w_{ij}^1\}$ is the maximum estimate on cost and scale factor α is selected from $\{0.01, 0.1, 0.5, 0.9\}$. The number of iterations for both baseline algorithms is set to 10 000.

The results of the comparisons on 1000 unseen instances are given in Table I. It can be found that both ACS and DRW are insensitive to the initial value τ_0 and always feasibly solve more than 700 instances. 63.51 % of the reduced networks filtered by the classifier model can be found a better solution by the decision model ($N = 1024$). Compared with the baseline algorithms, our PCD framework can provide better paths for more instances in a shorter time. Specifically, for the best baselines ACS ($\alpha = 0.5$) and DRW ($\alpha = 0.5$), the PCD ($N = 1024$) is superior to ACS in 926 instances and DRW in 919 instances in terms of solution quality (cost value) in all 1000 test instances. Another unique advantage of the framework is that the classifier model can provide interpretable information and terminate the solution process. In contrast, baseline algorithms cannot assert that a problem is unsolvable if they fail to find a feasible solution after some iterations.

V. Conclusions

In this work, we propose a learning-based solution framework for the complex constrained shortest path problems with multiple resource constraints. In the framework, the problem instance is first reduced in size by a preprocessing procedure, then a classifier model filters out the reduced networks that need to be further solved (i.e. there is a better path). Finally, a DRL-based heuristic attempts to construct the potential optimal path for the filtered networks. We also devise a crafted reward function and a shared low-variance baseline for the decision model optimization. The trained decision model can construct a better path in a short time due to limited sampling and fast inference. The experimental results show that the proposed framework outperforms two competitive heuristics and reaffirm that traditional iteration-based heuristics are not suitable for the problem with rare feasible solutions.

For the future studies, one can try to use reward shaping techniques to guide the behavior of the decision model towards reducing uncertainty about the environmental space and achieve better performance. Besides, attention can be paid to finding more effective node and edge representation learning modes, for example, trying to replace the traditional neighborhood aggregation method with representation learning based on path sequences in GNN learning, thereby improving classification and decision performance.

References

[1] L. D. P. Pugliese and F. Guerriero, "A survey of resource constrained shortest path problems: Exact solution approaches," *Networks*, vol. 62, no. 3, pp. 183–200, 2013.

[2] K. Li, Z. Gao, T. Tang, and L. Yang, "Solving the constrained shortest path problem using random search strategy," *Science China Technological Sciences*, vol. 53, no. 12, pp. 3258–3263, 2010.

[3] A. W. Mohemmed, N. C. Sahoo, and T. K. Geok, "Hybrid co-evolutionary particle swarm optimization and noising metaheuristics for the delay constrained least cost path problem," *Journal of Heuristics*, vol. 16, no. 4, pp. 593–616, 2010.

[4] Y. Marinakis, A. Migdalas, and A. Sifaleras, "A hybrid Particle Swarm Optimization -Variable Neighborhood Search Algorithm for Constrained Shortest Path Problems," *European Journal of Operational Research*, vol. 261, no. 3, pp. 819–834, 2017.

[5] X. Zhang, Y. Zhang, Y. Hu, Y. Deng, and S. Mahadevan, "An adaptive amoeba algorithm for constrained shortest paths," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7607–7616, 2013.

[6] H. Wang, X. Lu, X. Zhang, Q. Wang, and Y. Deng, "A Bio-Inspired Method for the Constrained Shortest Path Problem," *The Scientific World Journal*, vol. 2014, pp. 1–11, 2014.

[7] P. Avella, M. Boccia, and A. Sforza, "A penalty function heuristic for the resource constrained shortest path problem," *European Journal of Operational Research*, vol. 142, no. 2, pp. 221–230, 2002.

[8] F. A. Alghamdi, A. Y. Hamed, A. M. Alghamdi, A. B. Salah, T. H. Farag, and W. Hassan, "A genetic algorithm for shortest path with real constraints in computer networks." *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 13, no. 1, 2023.

[9] D. E. Golberg, "Genetic algorithms in search, optimization, and machine learning," *Addion wesley*, vol. 1989, no. 102, p. 36, 1989.

[10] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Springer Science & Business Media, 2006.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[12] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[13] J. W. Kim and S. K. Kim, "Fitness switching genetic algorithm for solving combinatorial optimization problems with rare feasible solutions," *The Journal of Supercomputing*, vol. 72, no. 9, pp. 3549–3571, 2016.

[14] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[15] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105400, 2021.

[16] Q. Wang and C. Tang, "Deep reinforcement learning for transportation network combinatorial optimization: A survey," *Knowledge-Based Systems*, vol. 233, p. 107526, 2021.

[17] N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman, "Learning combinatorial optimization on graphs: A survey with applications to networking," *IEEE access : practical innovations, open solutions*, vol. 8, pp. 120 388–120 416, 2020.

[18] W. Kool, H. van Hoof, and M. Welling, "Attention, Learn to Solve Routing Problems!" in *International Conference on Learning Representations*, 2019.

[19] I. Dumitrescu and N. Boland, "Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem," *Networks*, vol. 42, no. 3, pp. 135–153, 2003.

[20] L. Lozano and A. L. Medaglia, "On an exact method for the constrained shortest path problem," *Computers & Operations Research*, vol. 40, no. 1, pp. 378–384, 2013.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 2015, pp. 448–456.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," p. 11.

[24] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *International Conference on Learning Representations*, 2018.

[25] Z. Wang, J. Chen, and H. Chen, "EGAT: Edge-featured graph attention network," in *International Conference on Artificial Neural Networks*. Springer, 2021, pp. 253–264.

[26] X. Jiang, P. Ji, and S. Li, "CensNet: Convolution with Edge-Node Switching in Graph Neural Networks," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. Macao, China: International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 2656–2662.

[27] Z. Chen, J. Bruna, and L. Li, "Supervised community detection with line graph neural networks," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[28] Y. Yang and D. Li, "NENN: Incorporate Node and Edge Features in Graph Neural Networks," in *Proceedings of The 12th Asian Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. J. Pan and M. Sugiyama, Eds., vol. 129. PMLR, 2020, pp. 593–608.

[29] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[30] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," 2019.

[31] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 1263–1272.

[32] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[34] M. Dorigo and L. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[35] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.