AI Open 5 (2024) 46–54

Contents lists available at ScienceDirect

AI Open

journal homepage: www.keaipublishing.com/en/journals/ai-open

Full length article

Boosting graph search with attention network for solving the general orienteering problem

Zongtao Liu [a,*], Wei Dong [b], Chaoliang Wang [c], Haoqingzi Shen [c], Gang Sun [c], Qun jiang [c], Quanjin Tao [a], Yang Yang [a]

[a] Department of Computer Science and Technology, Zhejiang University, Hangzhou, 310013, China
[b] Huayun Information Technology Co., Ltd, Hangzhou, 310013, China
[c] Net Zhejiang Electric Power Co., Ltd. Marketing Service Center, Hangzhou, 311121, China

ARTICLE INFO

Keywords:
Orienteering problem
Beam search
Attention mechanism

ABSTRACT

Recently, several studies explore to use neural networks(NNs) to solve different routing problems, which is an auspicious direction. These studies usually design an encoder–decoder based framework that uses encoder embeddings of nodes and the problem-specific context to iteratively generate node sequence(path), and further optimize the produced result on top, such as a beam search. However, these models are limited to accepting only the coordinates of nodes as input, disregarding the self-referential nature of the studied routing problems, and failing to account for the low reliability of node selection in the initial stages, thereby posing challenges for real-world applications.

In this paper, we take the orienteering problem as an example to tackle these limitations in the previous studies. We propose a novel combination of a variant beam search algorithm and a learned heuristic for solving the general orienteering problem. We acquire the heuristic with an attention network that takes the distances among nodes as input, and learn it via a reinforcement learning framework. The empirical studies show that our method can surpass a wide range of baselines and achieve results iteratively generate the optimal or highly specialized approach.

1. Introduction

The orienteering problem(OP) is an important routing problem that originates from the sports game of orienteering. In this game, each competitor starts at a specific control point, visits control points in the tour, and will arrive at a destination point (usually the same as the start point). Each control point is associated with an associated prize, and the travel between control points involves a certain cost. The goal is to select a sequence of points such that the total prize is maximized within the cost constraint (Liu et al., 2021). This problem relates to several practical applications, such as resource delivery (Golden et al., 1987), tourist trip guide (Souffriau et al., 2008) and single-ring design in telecommunication networks (Thomadsen and Stidsen, 2003). Golden et al. (1987) have shown the OP to be an NP-hard problem, which motivates vigorous research into the design of heuristic solvers and approximation algorithms (Gunawan et al., 2016).

A recent trend in the machine learning community for solving routing problems is using deep neural networks to learn heuristic algorithms. With the help of other meta-algorithm such as beam search, some of these methods can achieve notable performance on several tasks (Li et al., 2018; Kool et al., 2018; Vinyals et al., 2015; Khalil et al., 2017; Nazari et al., 2018). These tasks include the traveling salesman problem(TSP), the orienteering problem(OP), and the vehicle routing problem(VRP). These studies deem the resolution of such problems as a sequence generation process from the coordinates of nodes, and use recurrent neural networks(RNNs) or graph neural networks(GNNs) to build a bridge between node patterns and the policy. Given that the OP falls within the routing problem domain, it aligns well with this methodology.

However, current learning-based algorithms have several limitations. Firstly, existing methods do not support direct input of travel cost information. They implicitly assume the travel cost, where the travel distance is the Euclidean distance between each pair of points, and take the coordinate of each position as network input to indirectly acquire the distance information. However, real-world distance metrics extend beyond Euclidean distance, limiting the practicality of such methods. For convenience, we here define the OP which only supports coordinates as input as the *coordinate orienteering problem*, and define the OP that supports direct input of costs among locations as the *general*

* Corresponding author.
E-mail address: zongtao.lzt@alibaba-inc.com (Z. Liu).

https://doi.org/10.1016/j.aiopen.2024.01.006
Received 30 November 2023; Received in revised form 16 January 2024; Accepted 30 January 2024
Available online 17 February 2024

orienteering problem. Secondly, the routing problems considered in the previous works possess self-referential property, but these works lack insight into it. For instance, in the orienteering problem, each step of point selection can be viewed as the initial step in another distinct orienteering problem, in which the competitor starts from the last selected points and will choose from the rest points. Lacking consideration about such property in the existing studies might degrade the utilization of samples when training.

Besides, most existing studies usually rely on a step-by-step beam search to acquire a better solution (Kaempfer and Wolf, 2018; Vinyals et al., 2015; Nowak et al., 2017); however, it might not work well on the routing problem with relative large problem size. The step-by-step beam search at each step prunes non-promising partial solutions based on a heuristic function and stores a fix-sized set (also called *beam*) of best alternatives. However, as the difficulty in the early stage of a routing problem is much greater than that in the later stage, the estimated heuristic score (probability or action value) to perform beam search in the initial step might also be less credible. To overcome this shortcoming, the stored partial solutions in the early stage of a search procedure should outnumber those in the later stage (since the less credible heuristic scores might mislead the pruning). How to design such a search algorithm is an important issue.

In this paper, we therefore propose a novel approach that combines a variant of the beam search algorithm with a learned heuristic to address the general orienteering problem. We acquire the heuristic with an attention network that takes both node attribute (prize) and edge attribute (cost) as input, and learn it via a reinforcement learning framework. Rather than formulating such routing tasks as sequence generation, we instead consider each step of point selection separately; that is, we view each state in the decision process as another different OP. This insight helps provide more direct reinforcement in each state and improves the utilization of training instances. The experimental result shows that our method can surpass a wide range of baselines and achieve results close to the optimal or highly specialized approach.

Accordingly, our contributions are as follows:

- We present a novel combination of an effective search algorithm and a learned heuristic for solving the general orienteering problem. This method can reduce the time and space complexity in the search process and offers flexibility in balancing the space, running time, and quality of plan.
- We designs an attention-based network that considers edge attributes to model routing problems. This is the first exploration to this type of problem that directly use edge attributes.
- By taking into account the self-referential property of such kind of problem, our method has better instances utilization compared with the previous encoder–decoder schemes.

Organization. The remainder of this paper is organized as follows. We first formulate the problem and review some relevant research. We then give the necessary definitions and introduce the proposed approach, including the search algorithm, the attention model, and the learning algorithm. Subsequently, we present the experimental results and finally conclude the work.

2. Related work

Orienteering problem. Over the years, many challenging applications in logistics, tourism, and other fields were modeled as OP. Meanwhile, quite a few studies for orienteering problems have been conducted. Golden et al. (1987) prove that the OP is NP-hard, i.e., no polynomial-time algorithm has been designed or is expected to be developed to solve this problem to optimality. Thus the targeted algorithms are very time-consuming, and for practical applications, heuristics will be necessary. Target solution methods (using branch-and-bound and branch-and-cut ideas) have been presented by Laporte and Martello (1990) and Ramesh et al. (1992). Heuristic approaches, which use traditional operation research techniques, have been developed by Tsiligirides (1984), Gendreau et al. (1998) and Tang and Miller-Hooks (2005).

Applications of neural networks in routing problems. Using neural networks (NNs) for optimizing decisions in routing problems is a long-standing direction, which can date back to the early work (Hopfield and Tank, 1985; Wang et al., 1995). These studies usually design an NN and learn a solution for each instance in an online fashion. Recently, several studies focus on using (D)NNs to learn about an entire class of problem instances. The critical point of these studies is to model permutation-invariant and variable-sized input appropriately.

Vinyals et al. (2015) introduce the Pointer Network(PN), which is inspired by the sequence-to-sequence model and uses attention to compute the conditional probability for variable-sized input, and train this model offline to solve the Coordinate TSP with the supervision by the optimal results. Bello et al. (2016) further introduces an Actor–Critic algorithm to train the PN. Nazari et al. (2018) replace the LSTM encoder of the PN by element-wise projections, which makes the encoder structure become permutation-invariant to the input nodes, and apply this model to address the vehicle routing problems (VRPs).

More recently, several studies leverage the power of GNNs that handling variable-sized and order-independent input to tackle the routing problems. Khalil et al. (2017) design an end-to-end framework to use the graph embeddings to estimate action value. This work can be viewed as the first exploration to use graph neural networks to solve routing problems. Nowak et al. (2017) train a graph neural network to directly output a tour as an adjacency matrix, and convert it into a feasible path by a beam search. Kool et al. (2018) train a Transformer (Vaswani et al., 2017a) based model to solve several variants of routing problems. However, these studies only focus on coordinating routing problems; thus the practicability is limited for real-world applications. Besides, current methods usually use a set-to-sequence/node-to-sequence model to perform routing, which ignores the self-referential property of many routing problems (including the OP).

Graph neural networks with node and edge information Models in the graph neural network (GNN) family have been successfully applied to many real-world applications, including protein interface prediction (Fout et al., 2017), neural machine translation (Vaswani et al., 2017b), relational extraction (Miwa and Bansal, 2016), and so on. Such success owes to the advances in their flexibility with various sized graphs, power of interpretation, training algorithms and so on. However, how to aggregate the rich information from both nodes and edges is still an open and long-standing question. Most existing GNN methods focus on graphs with the only node attributes or further consider the edge weight, which might have limited applications since real-world graphs usually have relational attributes (i.e., from edges). Recently, several works have explored this direction (Gong and Cheng, 2018; Beck et al., 2018; Schlichtkrull et al., 2017; Kim et al., 2019). For instance, Gong and Cheng (2018) propose a generalized graph attention mechanism to combine node and edge features, in which the multi-dimensional edge features are adjusted by both the local contents and the global layers. Beck et al. (2018) use an input transformation to convert the original graph into a bipartite one, in which edges are turned into special nodes and thus edges could also remain their hidden representations. However, these methods are all time-consuming for tackling the complete graph considered in our paper as input, thus designing an effective and efficient way to model our problem is challenging.

3. Problem definition

Formally, we define each orienteering problem instance s as a tuple with six elements $\langle V, C, R, T, v_{start}, v_{end} \rangle$. More specifically, $V =$

Algorithm 1 Cost-Level Beam Search

```
initialize a list of priority queues PQ[⌈T/τ⌉]
initialize the empty path P_g and insert it to PQ[0]
for t=0 to ⌈T/τ⌉ do
  while PQ[t] is not empty do
    pop partial path P_c from PQ[t]
    for v in (V − P_c) do
      pq = PQ[⌈cost(P_c + v)/τ⌉]
      if cost(P_c + v + v_end) >= T then continue
      insert (P_c + v) to pq
      if prize(P_c + v) > prize(P_g) then P_g = (P_c + p)
      if the size of pq is larger than K then
        pop argmin_P f(P, s) from pq
      end if
    end for
  end while
end for
return path P_g
```

$\{v_1, \ldots, v_M\}$ refers to the set of nodes (control points) containing the start node v_{start}, the prized nodes, and the end node v_{end}, where M denotes the number of nodes. C refers to the cost map where C_{v_i,v_j} represents the travel cost from v_i to v_j. R is the prize map where R_v indicates the prize collected when v is visited. Therefore, the problem is to find a path from $v_{start} \in V$ to $v_{end} \in V$ such that the total cost of the path is less than the prescribed cost limit T, and the overall prize collected from the nodes visited is maximized. The orienteering problem can also be formalized as a mixture integer problem, which can be referenced in Gunawan et al. (2016).

For notation convenience, here we introduce some operators or functions among these elements. $cost(P)$, $prize(P)$, and $last(P)$ represent the total cost, the total prize, and the last selected point of the (partial) selected path P, respectively. $(P + v_i)$ represents a selected path obtained by adding a new node v_i to the original path P. $(V - P)$ represents the node set that excludes the nodes in P from V.

4. Our approach

4.1. Cost-level beam search with the learned heuristic

Before introducing the search method proposed in this study, we first present an exact search method. We divide the maximal cost T into $\lceil T/\tau \rceil$ intervals with the length of τ. For each range, we maintain a queue to save all partial paths(solutions) with the total cost in that interval . Starting from interval $t = 0$, we iteratively retrieve the incomplete solution P from the front of the queue and scan all the nodes that are unselected. For each node v scanned, if $cost(P + v + v_{end})$ is no greater than the cost limit T, we then add the updated partial path $(P + v)$ to the queue in interval $(t + \lceil C_{last(P),v}/\tau \rceil)$. Finally, from all stored paths with the end node, the path with the highest total prize is the optimal path.

This method cannot finish computation in polynomial time. As the value of t increases, the number of stored partial paths per window increases exponentially. To reduce the space and time occupied by the search, at each step of the iteration, it is practical to prune some partial paths based on a predefined heuristic score $f(P, s)$ with the input of the current partial solution P and the problem instance s. To follow up this idea, we apply the idea of beam search and replace the queue maintained in each window t by a priority queue sized K, which is used to save the paths with the K highest heuristic scores in each window.

The total prize function $prize(\cdot)$ can be used as an intuitive instantiation of the heuristic score function f. Compared with the optimal solution, such pruning rule leaves the paths with K highest total prizes in each window, which significantly reduces the computational time and space. However, it is still difficult to avoid some short-sighted trajectories. In other words, the stored paths with top K total prizes in the current window do not necessarily overlap with the actual optimal or the near-optimal track. To reduce the possible impact of this defect in the search process, we introduce another heuristic score function f to estimate the total prize that can be reached along the partial path P:

$$f(P, s) = prize(P) + e(s'; \theta) \tag{1}$$

where f consists of two terms, in which the term $prize(P)$ computes the total prize of the given partial path, and the term $e(s'; \theta)$ is an estimation of the subsequent prize along the current path under π till the end of the problem. $e(\cdot\,; \theta)$ is an estimation function parameterized θ that computes the total potential prize given by an OP under the policy π_θ and s' represents the subproblem $\langle V - P, C, S, T - cost(P), last(P), v_{end} \rangle$ of the original OP.

A question that naturally arises here is how to obtain a reliable function $e(\cdot\,; \theta)$ to estimate the total prize of an orienteering problem. In the next section, we will present our solution with the attention-based neural networks learned by Q-learning (see Fig. 1).

4.2. Prize estimation via attention network

Kool et al. (2018) propose a network architecture based on self-attention to model several coordinate routing problems, including TSP, OP, and several VRP variants. With the benefit of reinforcement learning, the learned heuristic achieves better performance over other learning-based heuristic methods. Following the idea of using the attention mechanism to model the node interactions with permutation-invariant property, we propose an attention-based network, named **A**ttention **N**etwork (AN). Our proposed structure is not limited in handling coordinate OP, but can directly take all the node attributes (prize of each node), edge attributes (costs among nodes) and global attribute (the remaining cost of the agent) as input.

Edge-Aware Graph Attention Networks Layer. The graph attention networks(GAT) (Veličković et al., 2017) utilize the attention mechanism to aggregate the attributes of node neighbors adaptively, and extract high-level representation feature vector of each node, which can be formulated as follows:

$$\alpha_{vk} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \parallel \mathbf{W}\mathbf{h}_k]\right)\right)}{\sum_{j \in \mathcal{N}_v} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \parallel \mathbf{W}\mathbf{h}_j]\right)\right)} \tag{2}$$

$$\mathbf{h}'_{\mathcal{N}_v} = \sigma\left(\sum_{k \in \mathcal{N}_v} \alpha_{vk} \mathbf{W}\mathbf{h}_k\right) \tag{3}$$

$$MHA: \mathbf{h}'_{\mathcal{N}_v} = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{k \in \mathcal{N}_v} \alpha_{vk}^m \mathbf{W}^m \mathbf{h}_k\right) \tag{4}$$

where $h_v \in \mathbb{R}^H$ denotes the representation of node v, $\sigma(\cdot)$ denotes the sigmoid function, W and U are the learnable parameter matrices, and MHA is the abbreviation of multi-head average.

Since the original GAT can only aggregate the node information in the neighborhood of each node, but do not take into account the edge attributes. We modify the original structure of GAT; thus it can aggregate information from the connected edges, as well as the node neighbors and global information:

$$\alpha_{vk} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}[\mathbf{x}_v \parallel \mathbf{x}_k \parallel \mathbf{u}_{vk} \parallel \mathbf{g}]\right]\right)\right)}{\sum_{j \in \mathcal{N}_v} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}[\mathbf{x}_v \parallel \mathbf{x}_j \parallel \mathbf{u}_{vj} \parallel \mathbf{g}]\right]\right)\right)} \tag{5}$$

$$\mathbf{h}_{\mathcal{N}_v} = \sigma\left(\sum_{k \in \mathcal{N}_v} \alpha_{vk}\left[\mathbf{W}[\mathbf{x}_v \parallel \mathbf{x}_k \parallel \mathbf{u}_{vk} \parallel \mathbf{g}]\right]\right) \tag{6}$$

$$MHA: \mathbf{h}_{\mathcal{N}_v} = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{k \in \mathcal{N}_v} \alpha_{vk}^m\left[\mathbf{W}^m[\mathbf{x}_v \parallel \mathbf{x}_k \parallel \mathbf{u}_{vk} \parallel \mathbf{g}]\right]\right) \tag{7}$$

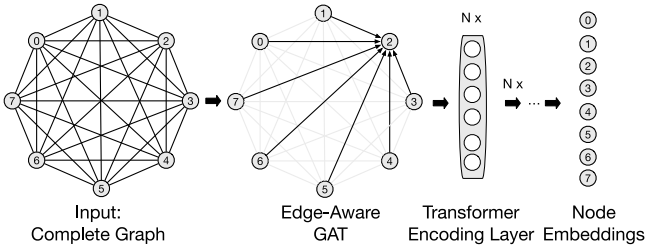


Fig. 1. An illustration of our attention network.

where x_v denotes v's node attribute (prize R_v), u_{vw} denotes the edge attribute (travel cost $C_{v,w}$) between node v and w, and g indicates the problem-level feature (the maximal cost T). After aggregating information from connected edges and node neighbors, we obtain the intermediate representation of each node $H = \{h_{v_1}, h_{v_i}, \ldots, h_{v_M}\}$, where h_v is the representation of v.

Transformer Encoding Layer(TEL). The Transformer model proposed in Vaswani et al. (2017a) has achieved advanced performance in several machine translation tasks, and has been used as a basic component for feature extraction of size/order-independent inputs in many models. Recently, Kool et al. (2018) also demonstrates the effectiveness of the Transformer to model many routing problems dealing with coordinates of points. Following this idea, we use the encoding layer of the Transformer in this study to extract the node features for the intermediate representation set $H = \{h_{v_1}, h_{v_2}, \ldots, h_{v_M}\}$. Since the resulting node representations are invariant to the input order, we do not use a positional encoding.

Each Transformer encoding layer has two sub-layers, the first of which is a multi-head self-attention mechanism followed by a node-wise fully connected network. The residual connection is used (He et al., 2016) around each of the two sub-layers, followed by layer normalization (Ba et al., 2016). To leverage these residual computations, the sub-layers in the encoder, as well as the embedding layers, produce outputs of dimension = 64. Because the use of the Transformer model is wide, and the implementation used in this paper is almost the same as the implementation in Vaswani et al. (2017a), we will omit the more detailed description of the structure of the Transformer encoding layer. Feeding the immediate embeddings H into N stacked Transformer encoding layers, we obtain the updated embeddings $H' = \{h'_{v_1}, h'_{v_2}, \ldots, h'_{v_M}\}$.

Linear Projection Layer. Finally, we use the linear combination of v_i's embedding h'_i to estimate the potential total prize r_i when selecting v_i in the next step:

$$q(s, v; \theta) = U h'_v \tag{8}$$

where $U \in \mathbf{R}^{|H|}$ is a learnable weight vector. Thus, we obtain the prize estimation $e(s;\theta) = max_{v \in V} q(s, v; \theta)$. Although it would be more convenient here to estimate the total prize of the given OP directly, we adopt this way for the sake of facilitating Q-learning to train the networks.

4.3. Training via Q-learning

We use a fitted Q-learning algorithm to obtain the potential future prize of each selection. Compared with fitting the optimal values generated by the exact approach like in Vinyals et al. (2015) and Li et al. (2018), the main advantage of this approach is that training with Q-learning can drastically save time spent, as the exact approach has very high time complexity. We formulate the states, actions, rewards, transitions, and policy in the reinforcement learning framework as follows:

States. As each step of the original OP $\langle V, C, S, T, v_{start}, v_{end}\rangle$ can be viewed as the first step of another new OP $\langle V - P, C, S, T - cost(P), last(P), v_{end}\rangle$, We define the state s as the current subproblem of the original OP.

Actions. The action v is a node selection of V that is in the part of the current state s. Both the definition of states and actions is applicable across node set with various sizes.

Rewards. We define the reward r of each action v as the prize collected when the action is visited.

Transitions. A transition is a tuple $\langle s_t, v_t, r_{t+1}, s_{t+1}\rangle$ which represents the agent takes action v_t in state s_t and then observes the immediate reward r_{t+1} and resulting state s_{t+1}.

Policy. Given the current state s, we can compute the q-value $q(s, v; \theta)$ of each unselected node v and apply a deterministic greedy policy $\pi(v|s) = \arg\max_{v \in |V|} q(s, v; \theta)$.

Training. We use double Q-learning to learn a greedy policy π_θ that is parameterized by the attention network and obtain the expected total prizes $q(s, v; \theta)$ of each action v simultaneously. The term "episode" in our context refers to a sequence of T steps involving node additions, beginning with an empty path and continuing until termination. Each individual step within an episode signifies a singular action of adding a node. The network parameters can be updated by minimizing the following loss function of the transitions picked up from the replayed memory:

$$\mathop{\mathbb{E}}_{s_t, a_t, s_{t+1} r_{t+1}}, \left[\left(y_t - q(s_t, v_t; \theta)\right)^2\right] \tag{9}$$

$$y_t = r_{t+1} - q(s_{t+1}, \arg\max_v q(s_{t+1}, v; \theta); \theta') \tag{10}$$

where θ' denotes parameters of the target Q-network updated periodically, and θ includes parameters of behavior Q-network outputting the action value for ϵ-greedy policy, same as the algorithm described in Van Hasselt et al. (2016). The motivation of adopting the double Q-learning instead of the vanilla Q-learning (Mnih et al., 2015) is that the vanilla Q-learning suffers from overestimations of action values and the double Q-learning reduces such overestimations. Besides, we do not introduce the decaying factor γ usually used in many reinforcement framework, because we need the estimated value to be more accurate. More details can be referenced in Algorithm 2.

4.4. Discussions

Comparison with the step-by-step beam search. The learned heuristic score function f of (1) can also be applied to the step-by-step beam search. However, as the routing in the early stage is much complicated than that in the later stage, the estimated heuristic score in the early stage might also be less reliable. In the initial stage of step-by-step beam search, the proportion of reliable selection could be very low and thus degrade the routing performance. Compared with the step-by-step beam search, the cost-level beam search store the early partial paths in the most extent.

Running times. As the computation of the estimated action values in each time window is parallelizable, the presented beam search algorithm can be significantly accelerated. The basic idea is to compute the action value on the GPU or TPU.

Algorithm 2 Double Q-learning for the Attention Network

Initialize replay memory D to capacity M
Initialize attention networks with random weights θ or pre-trained parameters
Initialize the problem instance $s_t^i \leftarrow new\ Instance(), \forall i \in \{1, .., B\}$
for count=1 to *max_iteration* **do**
  for t=1 to T_{max} **do**
    Choose node $v_t^i \leftarrow argmax_{v_t^i} q(s_t^i, v_t^i; \theta)$ with probability $1-\epsilon$, otherwise choose an action v^i randomly from the unselected nodes, $\forall i \in \{1, .., B\}$
    Obtain reward r_{t+1}^i and next state s_{t+1}^i, $\forall i \in \{1, .., B\}$
    Store the transition $\langle s_t^i, v_t^i, r_t^i, s_{t+1}^i \rangle$ in D, $\forall i \in \{1, .., B\}$
    Update state $s_t^i \leftarrow s_{t+1}^i$ if s_{t+1}^i is not a state of termination, otherwise $s_t^i \leftarrow new\ Instance(), \forall i \in \{1, .., B\}$
  end for
  Sample a batch of transitions $\{\langle s_t^i, v_t^i, r_t^i, s_{t+1}^i \rangle\}_{i=0}^{B'}$ from D
  Compute the target by (9)
  Compute gradients of θ and apply updates
end for

5. Experimental results

5.1. Dataset creation

To evaluate our proposed method against other algorithms, we generate problem instances with different settings. We sample each node v's coordinates x_v uniformly at random in the unit square and then compute the travel cost from node v to node w by their Euclidean distance $dis(v, w)$, where $dis(v, w)$ denotes the Euclidean distance between node v and w. We set the distance metrics as Euclidean distance because although many OP algorithms can handle the OP with different distance metrics, most released implementations can only support planar coordinates as input for convenience. Noting that selecting the Euclidean distance metric does not mean that we will compare the performance of our approach with those specialized methods designed for coordinate OP like (Khalil et al., 2017; Kool et al., 2018), because such comparison is unfair.

For the prize distribution, we consider three different settings described by Fischetti et al. (1998) and Kool et al. (2018). Besides, we normalize the prizes to make the normalized value fall between 0 and 1.

Constant. $r_v = 1$. Each node shares a fixed prize, so the agent aims to visit as many nodes as possible within the cost constraint.

Uniform. $r'_v \sim DiscreteUniform(1, 100); r_v = r'_v/100$. The prize of each node is discretized uniformed.

Distance. $r_v = (1 + \lfloor 99 \cdot \frac{dis(x_{v_{start}}, x_v)}{max_{v \in V} dis(x_{v_{start}}, x_v)} \rfloor)/100$. Each node has a (discretized) prize that is proportional to the distance to the start node, which is a challenging task as the node with the largest prize are the furthest away from the start node (Fischetti et al., 1998).

We choose the maximal travel cost T as a value close to half of the average optimal length of the corresponding TSP tour. Setting approximately the half of the nodes can be visited results in the most challenging problem instances (Vansteenwegen et al., 2011), as the number of possible node selections C_n^k is maximized if $k = n/2$. Finally, we set the value of T for the OP sized 20, 50, 100 as 2, 3, and 4 respectively, while the average length of the optimal TSP sized 20, 50, and 100 is 3.84, 5.70, and 7.76 respectively (Kool et al., 2018).

5.2. Model learning

The attention network proposed in Section 4 is trained via double Q-learning, as illustrated in Algorithm 2. For the hyperparameters of the network, we set the hidden size as 32, the number of multi-heads in both variant-GAT and Transformer as 8 and the number of stacked Transformer encoding layers as 4. For the hyperparameters of q-learning, we set the size of replay memory as 1024, and use Adam optimizer (Kingma and Ba, 2015) with the learning rate of 1×10^3 to update the model parameters. The number of training instances is 100 K and the maximal iteration is set as 20 K.

5.3. Performance comparison

Comparable Methods. We evaluate the performance of our method under the above-mentioned instance generation settings . Firstly, we compare our proposed method with an exact method and a state-of-the-art heuristic method:

- *Gurobi:* This baseline is computed by the Gurobi optimizer (Optimization, 2014), which solves the mix integer problem corresponding to each instance and produces the optimal results. It is intractable for problem size more than 20.
- *Compass:* This is a Genetic Algorithm for solving the OP (Kobeaga et al., 2018).

In addition, we further compare with several policy-based methods, which compute the node selection probability $p(v|s)$ or action value $q(s, v)$ in state s. We report their results based on two meta-algorithm: (1)greedy: the node with the highest probability or action value is selected in each state. (2) beam: we apply a step-by-step beam search for some of the below methods to obtain the target path. The beam size is set as 100.

- *Random:* At each state, a node is randomly selected from the rest nodes.
- *Tsili:* Tsiligirides (1984) propose a manual defined function to compute node selection probability in each state.
- *AN-dqn:* At each step, the node v with the highest action value $q(s, v; \theta)$ will be selected. The beam search version of this baseline is the step-by-step beam search with the learned heuristic.
- *AN-a2c:* We apply the advantage Actor–Critic algorithm to learn the policy of each state. For each state s, an attention network proposed in Section 4 is shared for the actor-network and the critic-network to obtain node embeddings. In the actor-network, we then project the embedding of each node to get the logit by a linear combination, followed by a softmax operator to get the normalized selection probability. In the critic-network, we map the sum of embeddings to the estimated value by another linear layer.
- *AN-at:* Kool et al. (2018) propose an attention-based encoder–decoder architecture to solve the routing problem and train it using policy gradient with the deterministic greedy rollout baseline. As the original encoder used in Kool et al. (2018) can only support the input of nodes' coordinates, we replace it by the attention network proposed in Section 4.
- *AN-pt:* Nazari et al. (2018) design an encoder–decoder based model to perform routing by decoding the encoded embeddings of nodes via pointer network (Vinyals et al., 2015). As the original encoder used in Nazari et al. (2018) can only support nodes' coordinates as input, we use our attention network as the encoder.

In addition to the above methods serving as benchmark approaches, the comparison between $AN - a2c$, $AN - at$, and $AN - pn$ also demonstrates the effectiveness of our network architecture compared with the encoder–decoder schemes used in the previous works. This is because $AN - a2c$ can be view as a policy gradient version of our proposed DQN framework proposed in Section 4.3, which considers each step of node selection as a distinct OP. In contract, $AN - at$ and $AN - pn$ view each selection in connection with the previous ones (via RNN or self-attention mechanism).

Table 1
Performance comparison. The gap % is w.r.t. the best value across all methods.

Prize	Method	$n = 20, T = 2$			$n = 50, T = 3$			$n = 100, T = 4$		
		Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Constant	Gurobi	**10.57**	0.00%	(1 s)	–			–		
	Compass	10.56	0.09%	(0 s)	**29.58**	0.00%	(2 s)	**59.35**	0.00%	(4 s)
	Random	4.20	60.26%	(0 s)	6.54	77.89%	(0 s)	8.61	85.49%	(0 s)
	Tsili (greedy)	8.82	16.56%	(0 s)	23.89	19.24%	(5 s)	47.65	19.71%	(5 s)
	AN-dqn (greedy)	9.52	9.93%	(0 s)	25.20	14.81%	(0 s)	49.74	16.19%	(1 s)
	AN-a2c (greedy)	9.20	12.96%	(0 s)	18.20	38.47%	(0 s)	44.60	24.85%	(1 s)
	AN-at (greedy)	6.87	35.00%	(0 s)	11.75	60.28%	(0 s)	17.69	70.19%	(1 s)
	AN-pn (greedy)	7.75	26.68%	(0 s)	15.38	48.01%	(0 s)	17.55	70.43%	(1 s)
	Tsili (beam)	10.01	9.62%	(0 s)	24.58	28.75%	(2 s)	47.38	20.17%	(9 s)
	AN-dqn (beam)	10.38	9.59%	(1 s)	29.04	1.82%	(10 s)	56.70	4.46%	(39 s)
	AN-a2c (beam)	9.76	7.66%	(1 s)	27.86	5.81%	(5 s)	50.10	15.59%	(24 s)
	AN-at (beam)	8.28	21.67%	(1 s)	17.24	41.72%	(5 s)	37.69	36.50%	(22 s)
	AN-pn (beam)	8.60	18.64%	(1 s)	13.69	53.72%	(4 s)	21.12	64.41%	(19 s)
	CS-l (20, 0.05)	10.43	1.32%	(1 s)	28.14	4.87%	(3 s)	55.71	6.13%	(14 s)
	CS-p (20, 0.05)	**10.57**	0.00%	(1 s)	29.16	1.42%	(6 s)	57.11	3.77%	(44 s)
Uniform	Gurobi	**5.85**	0.00%	(1 s)	–			–		
	Compass	5.84	0.17%	(0 s)	**16.46**	0.00%	(2 s)	**33.30**	0.00%	(6 s)
	Random	2.13	63.59%	(0 s)	3.33	79.77%	(0 s)	4.37	86.88%	(0 s)
	Tsili (greedy)	4.85	17.09%	(0 s)	12.46	22.94%	(0 s)	25.48	23.48%	(0 s)
	AN-dqn (greedy)	5.15	11.97%	(0 s)	13.91	15.49%	(0 s)	26.51	20.39%	(1 s)
	AN-a2c (greedy)	4.97	15.04%	(0 s)	14.13	14.16%	(0 s)	24.31	27.00%	(1 s)
	AN-at (greedy)	3.98	31.97%	(0 s)	8.22	50.06%	(0 s)	9.75	70.72%	(1 s)
	AN-pn (greedy)	4.46	23.76%	(0 s)	11.10	32.56%	(1 s)	10.06	69.79%	(2 s)
	Tsili (beam)	5.54	5.30%	(0 s)	13.54	15.63%	(2 s)	25.91	22.19%	(9 s)
	AN-dqn (beam)	5.74	18.80%	(1 s)	15.76	4.25%	(10 s)	30.61	8.07%	(40 s)
	AN-a2c (beam)	5.57	4.79%	(1 s)	15.13	8.08%	(5 s)	27.40	17.72%	(23 s)
	AN-at (beam)	4.45	23.93%	(1 s)	9.45	16.46%	(4 s)	10.55	68.32%	(20 s)
	AN-pn (beam)	4.84	17.26%	(1 s)	13.75	42.59%	(4 s)	12.08	63.72%	(24 s)
	CS-l (20, 0.05)	5.75	1.71%	(1 s)	15.42	6.32%	(3 s)	30.53	8.32%	(12 s)
	CS-p (20, 0.05)	5.82	0.51%	(1 s)	16.13	2.00%	(7 s)	30.93	7.12%	(40 s)
Distance	Gurobi	**5.39**	0.00%	(4 s)	–			–		
	Compass	5.37	0.37%	(0 s)	**16.17**	0.00%	(3 s)	**33.19**	0.00%	(8 s)
	Random	1.89	64.94%	(0 s)	2.90	82.07%	(0 s)	3.81	88.52%	(0 s)
	Tsili (greedy)	4.08	24.30%	(0 s)	12.46	22.94%	(0 s)	25.69	22.60%	(0 s)
	AN-dqn (greedy)	4.77	11.50%	(0 s)	13.70	15.28%	(0 s)	26.60	19.86%	(1 s)
	AN-a2c (greedy)	4.57	15.21%	(0 s)	13.70	15.28%	(0 s)	22.32	32.75%	(1 s)
	AN-at (greedy)	3.62	32.84%	(0 s)	9.29	42.55%	(1 s)	10.52	68.30%	(2 s)
	AN-pn (greedy)	3.75	30.43%	(0 s)	11.37	29.68%	(1 s)	10.77	67.55%	(2 s)
	Tsili (beam)	5.26	1.68%	(0 s)	15.50	14.50%	(2 s)	30.53	8.04%	(10 s)
	AN-dqn (beam)	5.27	11.5%	(1 s)	15.82	4.14%	(9 s)	30.28	8.77%	(35 s)
	AN-a2c (beam)	5.02	6.86%	(1 s)	15.13	6.43%	(6 s)	27.83	16.15%	(24 s)
	AN-at (beam)	4.42	18.00%	(1 s)	10.68	33.95%	(4 s)	11.10	66.56%	(21 s)
	AN-pn (beam)	4.24	21.34%	(1 s)	12.15	24.86%	(4 s)	12.48	62.40%	(17 s)
	CS-p (20, 0.05)	5.31	1.48%	(1 s)	15.58	3.65%	(3 s)	30.77	7.29%	(13 s)
	CS-l (20, 0.05)	5.35	0.74%	(1 s)	15.99	1.11%	(6 s)	31.16	6.12%	(42 s)

Lastly, we report the performance of our cost-level beam search algorithm with different heuristic function and settings:

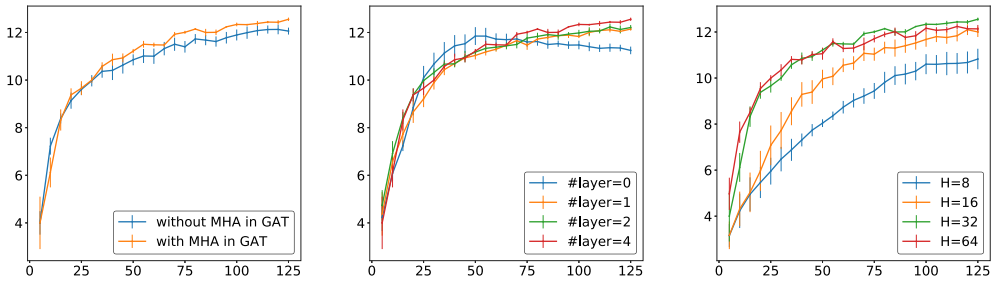
- *CS-p (K, τ):* This approach refers to the cost-level beam search with $prize(\cdot)$ as the heuristic score function. K and τ represent the beam size and cost interval, respectively.
- *CS-l (K, τ):* This is our proposed method, which represents the cost-level search with the learned heuristic score function.

Comparison Results. Table 1 reports the performance and average running time of each approach on 10 K test instances. Presenting running time is not for the comparison between the efficiency of different methods. The results of different approaches might not be comparable due to the differences in their hardware usages (e.g., CPU or GPU), details of implementation (e.g., C++/Python with/without some optimizations), and the settings of hyperparameters (e.g., beam size). We report the time spent mainly to show that in general, our method and implementation are time-affordable with the help of GPU acceleration.

We note that our search method with the learned heuristic ($CS-l$) surpasses all comparative methods except the exact approach *Gurobi* and the specialized genetic algorithm *Compass* (the gap is relatively small compared with those of other benchmarks), which demonstrates the effectiveness of our proposed methods. *AN-a2c* is a competitive method using other baselines with attention network as a shared bottom. However, since the step-by-step beam search retains relatively fewer partial paths at the start of the selection process, our search method significantly outperforms this variant. We also observed that by introducing the learned heuristic score function, our cost-level beam search method could achieve better performance than the search with the manual defined function. This shows the importance of using the learned function.

The $AN-dqn(beam)$ can be viewed as the step-by-step variant of our proposed method ($CS-l$). We observe that our approach can achieve better performance than $AN-dqn(beam)$, which demonstrates the effectiveness of the cost-level beam search.

Comparing the results of *AN-a2c*, *AN-at*, and *AN-pn* in a greedy fashion or a beam search fashion, we can find which scheme of network



(a) With/without MHA in GAT. (b) Varing #layer of TLE. (c) Varing hidden size.

Fig. 2. Model parameter/structure analysis. The x-axis is the number of episodes. The y-axis denotes the average total prize on 200 test instances with greedy policy using the estimated action values from AN. This experiment is conducted five times, and we show the average performance and the corresponding standard error.

Table 2
Generalization analysis. We list the performance of $CS-l$ with the heuristic trained on different problem size (20/50/100). The prize type is set as *distance*, while the testing problem size is set as 100.

Problem size in training	20	50	100
Performance in testing	30.38	30.85	31.16

design works better for the OP. We find that *An-a2c* significantly outperforms other two approaches. This shows it is necessary to take into account the self-referential property of the OP and use a non-sequential network to model this problem.

5.4. Parameter/structure analysis

In this section, we explore how the hyper-parameters or the structure design influence the quality of modeling the OP by our attention network. To achieve this goal, we present the performance of AN-dqn (greedy policy using DQN) on 200 instances in the learning process. Each instance has 50 prized nodes, and the prize of each node is proportional to its distance to the start node (*distance*). We first explore to what extent the multi-head used in the GAT improves the performance, which is shown in Fig. 2(a). We note that by introducing multi-head, the performance increases, which shows that the multi-head helps the GAT better aggregate edge information. Following the GAT, the Transformer encoding layer(TEL) further extracts more high-level representations for nodes. From Fig. 2(b), we observe that the performance improves as the number of layer increases. Therefore, adding the Transformer encoding layer is helpful for modeling the OP. Lastly, we explore the relationship between the hidden size $\mathbb{R}^H$ and the quality of the learned model. We note that a larger hidden size leads to the better performance.

5.5. Generalization analysis

To evaluate the capability of generalization of our method, we test the generalization performance on different problem sizes. We report results on 10 K instances with the prize type as *distance* (Cf. Table 2), because *distance* is considered the hardest of these prize types. The performance drop is 2.5% with the heuristic trained for the size of 20, and 0.99% with the heuristic trained for the size of 50. The performance change is tiny but still outperforms the most in Table 1, which shows that our method has a good capability of generalization.

5.6. Visualization

Finally, we visualize the output paths generated by $AN-dqn$, $CS-p$, and $CS-l$ with the problem size of 50 and the prize type of *distance* in Fig. 3. We observe that compared with the results of the greedy selection with action values ($AN-dqn$) or the cost-level beam search with a simple heuristic function ($CS-p$), the combination of our cost-level beam search and the learned heuristic ($CS-l$) can conduct more look-forwarding selection, thus have better performance.

6. Conclusion and future work

To what extent can we combine the deep learning models and the heuristic search methods to get better routing solutions? In this paper, we propose a novel combination of a cost-level beam search method and the learned heuristic score function to solve the NP-hard problem of the general orienteering problem. To estimate the potential prize of the current partial solution, we design an attention-based network. It first uses a variant GAT to aggregate both the edge attribute(travel cost) and the node attribute(prize), then applies stacked Transformer encoding layers to produce high-level node representations. Rather than modeling the OP in an encoder–decoder structure, we use the extracted node representations to directly estimate the target action values, which is a better scheme for the OP with the self-referential property. We further conduct sufficient experiments to show that our method outperforms most of the benchmark methods and obtains the comparable results to the exact method and the highly optimized and specialized method. We also note that the performance of our method has been improved by introducing the deep neural network.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Wei Dong is currently employed by Huayun Information Technology Co., Ltd. Chaoliang Wang, Haoqingzi Shen, Gang Sun and Qun Jiang are currently employed by Net Zhejiang Electric Power Co., Ltd.

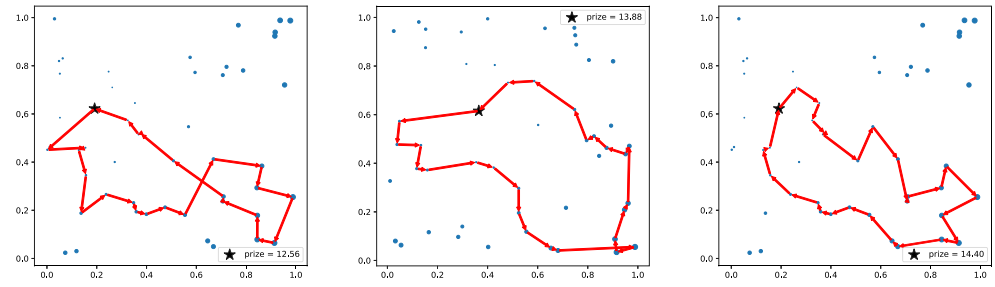
Appendix A. Sample appendix section

In this section, we present the necessary details about our empirical studies. Our experiment is conducted on a single machine with an Intel Xeon E5 and an NVIDIA TITAN GPU. We implement our neural networks by Pytorch.

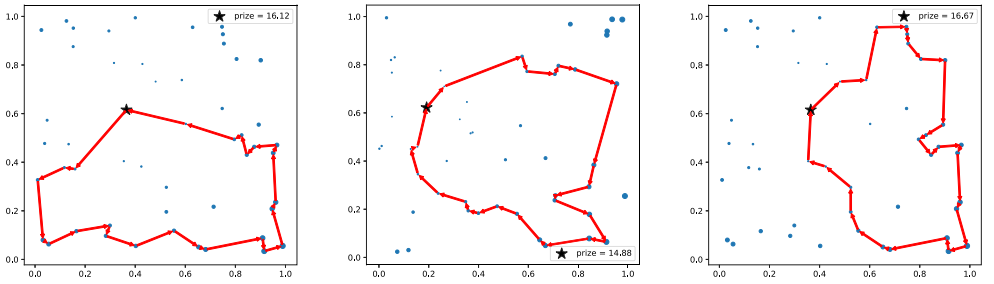
Details of baselines

In this subsection, we present the implementation details of some of the baselines in Section 5.3.

Hyperparameters. For the implementations of the NN-based baselines, we set the same hyperparameters. In the experiment, we conduct the training procedure 5 times and report the result from the model with the best performance on 200 validation instances (see Table A.3).



(a) result of AN-dqn on Instance 1 (b) result of AN-dqn on Instance 2 (c) result of CS-p on Instance 1



(d) result of CS-p on Instance 2 (e) result of CS-l on Instance 1 (f) result of CS-l on Instance 2

Fig. 3. Visualization of the output paths by AN-dqn, CS-p and CS-l with the problem size of 50 and the prize type of *distance*. The star marker denotes the start/end node, and the circle marker represents the prized node, the area of which is proportional to its prize.

Table A.3
Hyperparameters for NNs.

Hyperparameter	Value	Hyperparameter	Value
#head in GAT	20	#head in TEL	8
#TEL	4	Hidden size	64
Learning rate	1e−3	Optimizer	Adam

Beam Search. For the beam search versions of $Tsili, AN-a2c, AN-at, AN-pn$, we set the heuristic function as the accumulation of log-likelihood probabilities $\sum_i logp_i$. For the beam search version of $AN-dqn$, we set the heuristic function the same as (1).

Tsili. The heuristic function to compute $p(v|s)$ used by Tsiligirides (1984) is a formula with multiple terms to compute the probability that a node should be expanded, but by tuning the weights the form with only one simple term works best, showing the difficulty of manually defining a good probability distribution. We use the following formula to compute such probability:

$$f(v,P) = \left(\frac{R_v}{C_{last(P),v}}\right)^4. \tag{A.1}$$

Let S be the set with $\min(4, n-(t-1))$ unvisited nodes with maximum score f_v. Then the node probabilities p_v at time t are defined as

$$p_v = p_\theta(v|s) = \begin{cases} \frac{f_v}{\sum_{w\in S} f_w} & \text{if } v \in S \\ 0 & \text{otherwise.} \end{cases} \tag{A.2}$$

AN-a2c. We apply the advantage Actor–Critic algorithm to learn the policy of each state. For each state s, an attention network proposed in Section 4 is shared for the actor-network and the critic-network to obtain node embeddings $\{h_{v_1}, h_{v_1}, \dots, h_{v_M}\}$. In the actor network, we then project the embedding of each node to get the logit by a linear combination, followed by a softmax operator to get the normalized selection probability $p(v|s) = softmax(U \cdot h_v)$. In the critic network, we map the sum of embeddings to the estimated value by another linear layer, $V(s) = W \cdot \sum_v h_v$. U and V are learnable variables.

For notation convenience, we define θ_1 as the parameters of actor network, and θ_2 as the parameters of critic networks. The loss function is set as:

$$loss(\theta_1) = \mathbb{E}[-log(p(v_t|s_t))(V(s_t) - V(s_{t+1}) - r_{t+1})] \tag{A.3}$$

$$loss(\theta_2) = \mathbb{E}[(V(s_t) - V(s_{t+1}) - r_{t+1}^2] \tag{A.4}$$

$$loss(\theta_1, \theta_2) = loss(\theta_1) + 0.5 \cdot loss(\theta_2) \tag{A.5}$$

AN-at . Kool et al. (2018) propose an attention-based encoder–decoder architecture to solve the routing problem and train it using policy gradient with the deterministic greedy rollout baseline. As the original encoder used in Kool et al. (2018) can only support the input of nodes' coordinates, we replace it by the attention network proposed in Section 4.

More specifically, in the decoding step, we set the context for the decoder input at step t is the current/last location $last(P)$ and the remaining travel cost $T - cost(P)$. The context is defined as:

$$\mathbf{h}_{(c)}^{(N)} = \begin{cases} [\bar{\mathbf{h}'}, \mathbf{h}'_{last(P)}, T - cost(P)] & t > 1 \\ [\bar{\mathbf{h}'}, \mathbf{h}'_0, T - cost(P)] & t = 1. \end{cases} \tag{A.6}$$

where $\mathbf{h}'_v$ denotes the node v's embedding, $\bar{\mathbf{h}'}$ represents the average node embedding and $\mathbf{h}'_0$ is a learnable vector as the decoder initial input. All visited nodes in the decoding are masked.

AN-pn. Nazari et al. (2018) design an encoder–decoder based model to perform routing by decoding the encoded embeddings of nodes via pointer network (Vinyals et al., 2015). As the original encoder used in Nazari et al. (2018) can only support nodes' coordinates as input, we use our attention network as the encoder.

At decoder step i, we use a context-based attention mechanism, which extracts the relevant information from the inputs using a variable-length alignment vector a_i.

We let $\mathbf{h}'_{\mathbf{t,v}}$ be the embedded input of node v at step t, and $h_t \in \mathbb{R}^D$ be the hidden state of the LSTM cell at decoding step t. The alignment vector a_t is then computed by

$$a^t = a_t(h'_{t,v}, h_t) = \text{softmax}\left(u_t\right) \tag{A.7}$$

$$u_t^i = v_a^T \tanh\left(W_a[\bar{h'}_{t,v} \parallel h_t]\right) \tag{A.8}$$

We obtain the conditional probabilities by combining the context vector c_t, as follows:

$$c_t = \sum_{i=1}^{M} a_t^i \bar{h'}_{t,v}, \tag{A.9}$$

with the embedded inputs, and then use softmax function to normalize the values:

$$P(v_t|s_t) = \text{softmax}(\tilde{u}_t^i) \tag{A.10}$$

$$\tilde{u}_t^i = v_c^T \tanh\left(W_c[\bar{h'}_{t,v} \parallel c_t]\right). \tag{A.11}$$

In (A.7)–(A.11), v_a, v_c, W_a and W_c are trainable variables.

References

Ba, J.L., Kiros, J.R., Hinton, G.E., 2016. Layer normalization. arXiv preprint arXiv:1607.06450.

Beck, D., Haffari, G., Cohn, T., 2018. Graph-to-sequence learning using gated graph neural networks. In: Gurevych, I., Miyao, Y. (Eds.), Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers. Association for Computational Linguistics, pp. 273–283. http://dx.doi.org/10.18653/v1/P18-1026, URL: https://www.aclweb.org/anthology/P18-1026/.

Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S., 2016. Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940.

Fischetti, M., Gonzalez, J.J.S., Toth, P., 1998. Solving the orienteering problem through branch-and-cut. INFORMS J. Comput. 10 (2), 133–148.

Fout, A., Byrd, J., Shariat, B., Ben-Hur, A., 2017. Protein interface prediction using graph convolutional networks. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. pp. 6530–6539, URL: http://papers.nips.cc/paper/7231-protein-interface-prediction-using-graph-convolutional-networks.

Gendreau, M., Laporte, G., Semet, F., 1998. A tabu search heuristic for the undirected selective travelling salesman problem. European J. Oper. Res. 106 (2–3), 539–545.

Golden, B.L., Levy, L., Vohra, R., 1987. The orienteering problem. Nav. Res. Logist. 34 (3), 307–318.

Gong, L., Cheng, Q., 2018. Adaptive edge features guided graph attention networks. CoRR abs/1809.02709. arXiv:1809.02709. URL: http://arxiv.org/abs/1809.02709.

Gunawan, A., Lau, H.C., Vansteenwegen, P., 2016. Orienteering problem: A survey of recent variants, solution approaches and applications. European J. Oper. Res. 255 (2), 315–332.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778.

Hopfield, J.J., Tank, D.W., 1985. "Neural" computation of decisions in optimization problems. Biol. Cybern. 52 (3), 141–152.

Kaempfer, Y., Wolf, L., 2018. Learning the multiple traveling salesmen problem with permutation invariant pooling networks. arXiv preprint arXiv:1803.09621.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L., 2017. Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems. pp. 6348–6358.

Kim, J., Kim, T., Kim, S., Yoo, C.D., 2019. Edge-labeling graph neural network for few-shot learning. CoRR abs/1905.01436. arXiv:1905.01436. URL: http://arxiv.org/abs/1905.01436.

Kingma, D.P., Ba, J., 2015. Adam: A method for stochastic optimization. In: ICLR.

Kobeaga, G., Merino, M., Lozano, J.A., 2018. An efficient evolutionary algorithm for the orienteering problem. Comput. Oper. Res. 90, 42–59.

Kool, W., van Hoof, H., Welling, M., 2018. Attention, learn to solve routing problems! arXiv preprint arXiv:1803.08475.

Laporte, G., Martello, S., 1990. The selective travelling salesman problem. Discrete Appl. Math. 26 (2–3), 193–207.

Li, Z., Chen, Q., Koltun, V., 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In: Advances in Neural Information Processing Systems. pp. 539–548.

Liu, Z., Xu, J., Su, J., Xiao, T., Yang, Y., 2021. Boosting graph search with attention network for solving the general orienteering problem. ArXiv abs/2109.04730, https://api.semanticscholar.org/CorpusID:237485319.

Miwa, M., Bansal, M., 2016. End-to-end relation extraction using LSTMs on sequences and tree structures. CoRR abs/1601.00770. arXiv:1601.00770. URL: http://arxiv.org/abs/1601.00770.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.

Nazari, M., Oroojlooy, A., Snyder, L., Takác, M., 2018. Reinforcement learning for solving the vehicle routing problem. In: Advances in Neural Information Processing Systems. pp. 9839–9849.

Nowak, A., Villar, S., Bandeira, A.S., Bruna, J., 2017. A note on learning algorithms for quadratic assignment with graph neural networks. Statistics 1050, 22.

Optimization, G., 2014. Inc.,"Gurobi optimizer reference manual," 2015.

Ramesh, R., Yoon, Y.-S., Karwan, M.H., 1992. An optimal algorithm for the orienteering tour problem. ORSA J. Comput. 4 (2), 155–165.

Schlichtkrull, M.S., Kipf, T.N., Bloem, P., van den Berg, R., Titov, I., Welling, M., 2017. Modeling relational data with graph convolutional networks. CoRR abs/1703.06103. arXiv:1703.06103. URL: http://arxiv.org/abs/1703.06103.

Souffriau, W., Vansteenwegen, P., Vertommen, J., Berghe, G.V., Oudheusden, D.V., 2008. A personalized tourist trip design algorithm for mobile tourist guides. Appl. Artif. Intell. 22 (10), 964–985.

Tang, H., Miller-Hooks, E., 2005. A tabu search heuristic for the team orienteering problem. Comput. Oper. Res. 32 (6), 1379–1407.

Thomadsen, T., Stidsen, T.K., 2003. The Quadratic Selective Travelling Salesman Problem. Informatics and Mathematical Modeling Technical Report.

Tsiligirides, T., 1984. Heuristic methods applied to orienteering. J. Oper. Res. Soc. 35 (9), 797–809.

Van Hasselt, H., Guez, A., Silver, D., 2016. Deep reinforcement learning with double q-learning. In: Thirtieth AAAI Conference on Artificial Intelligence.

Vansteenwegen, P., Souffriau, W., Van Oudheusden, D., 2011. The orienteering problem: A survey. European J. Oper. Res. 209 (1), 1–10.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017a. Attention is all you need. In: NIPS. pp. 5998–6008.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017b. Attention is all you need. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. pp. 5998–6008, URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., 2017. Graph attention networks. arXiv preprint arXiv:1710.10903.

Vinyals, O., Fortunato, M., Jaitly, N., 2015. Pointer networks. In: Advances in Neural Information Processing Systems. pp. 2692–2700.

Wang, Q., Sun, X., Golden, B.L., Jia, J., 1995. Using artificial neural networks to solve the orienteering problem. Ann. Oper. Res. 61 (1), 111–120.