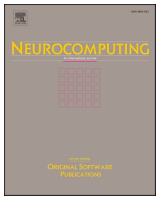
Neurocomputing 579 (2024) 127392

Contents lists available at ScienceDirect

Neurocomputing

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



G-DGANet: Gated deep graph attention network with reinforcement learning for solving traveling salesman problem

Getu Fellek *, Ahmed Farid, Shigeru Fujimura, Osamu Yoshie, Goytom Gebreyesus

Graduate School of Information, Production and Systems, Waseda University, Kitakyushu City, Japan

ARTICLE INFO

Keywords:
Graph neural network
Deep reinforcement learning
Traveling salesman problem
Deep graph representation learning

ABSTRACT

Combinatorial optimization problem (COP) is an NP-hard problem for which finding an optimal solution is difficult, especially as the problem size increases. The Traveling Salesman Problem (TSP), one of the COPs that can be formulated over a graph, is a well-researched area in operations research and computer science. Deep Reinforcement Learning (DRL) is now regarded as a promising approach for solving TSP and other NP hard problems. In this paper, we propose a novel Gated Deep Graph Attention Network (G-DGANet) which builds upon the existing Graph Neural Network (GNN) to solve TSP. The proposed G-DGANet uses gating mechanism between subsequent layers of the network to extract representations of nodes deeper in the network without loss in performance. G-DGANet also designs a novel aggregator to construct global graph embeddings from different embedding preferences. In addition, to effectively learn underlying structure of a graph, G-DGANet integrates node and edge information of the graph while updating node representations in the message passing mechanism. We used proximal policy optimization (PPO) to train G-DGANet on randomly generated instances. We conducted an experiment on randomly generated instances and on real-world road network data generated from digital maps to verify the performance of G-DGANet. The findings from experiments demonstrate that G-DGANet outperforms most traditional heuristics and existing DRL approaches, with high generalization abilities from random instance training to random instance testing and real-world road network instance testing.

1. Introduction

Combinatorial optimization problem (COP) is an NP-hard problem for which finding an optimal solution is difficult, especially as the problem size increases. It is a highly practical and extensively researched field in computer science and operations research with several applications in areas including social networks, transportation, logistics, telecommunications, and planning. Its primary goal is to produce an optimal solution for a specific problem instance while obeying a given set of constraints [1–3]. Traveling Salesman Problem (TSP) is one of the combinatorial optimization problems that optimizes the total time to visit a sequence of customer nodes exactly once by starting and ending the trip at the same node in the sequence. With the coordinates of node locations as a 2D vertices and the Euclidean distance between customer nodes as edge, TSP can be represented on graph [4].

Classical solution approach of TSP and other COPs comprises of heuristic [5], approximate [6] and exact methods. However, these approaches have inherent drawbacks that make them ineffective in solving large-scale problems. For example, heuristics solutions necessitate extensive domain knowledge for frequent hand engineering. Similarly, due to computational complexity, exact solutions do not solve large scale instances, and approximate solutions have a weak optimality guarantee [7].

Recently, following the increasing impact of Artificial intelligence, deep learning-based methods are achieving promising results. Deep learning has made it possible to automate the design of heuristics for improved solution quality of COPs. The emergence of Pointer networks (PtrNet) [8] as a pioneering work for using deep learning to solve TSP is marked as fundamental transition from conventional solution to deep learning-based approaches to address COPs. By using an attention-based sequence to sequence model, a supervised learning based PtrNet generates a tour for TSP as a permutation of the input sequence. Following PtrNet several other deep learning models are being used to address TSP. According to existing literatures, these approaches are mainly based on encoder decoder framework using Recurrent Neural Network (RNN), Graph Neural Network (GNN) and Transformer based networks [1], [9–11]. However, unlike PtrNet, owing to the difficulty of obtaining a labeled data, Reinforcement Learning is used in place of supervised

* Corresponding author.
E-mail address: getutadesse@toki.waseda.jp (G. Fellek).

https://doi.org/10.1016/j.neucom.2024.127392
Received 7 October 2022; Received in revised form 27 September 2023; Accepted 6 February 2024
Available online 15 February 2024

learning to solve COPs effectively [12] [13]. In the encoder-decoder framework-based solution generation, an encoder encodes raw features into higher dimensional embedding and the decoder processes the features from encoder for sequential decoding in the construction of an optimal tour.

In a GNN based encoders in particular, non-Euclidean information on the graph is encoded into continuous nodes representation using graph embedding to capture the topological relations between nodes of a graph. After GNN based graph representation learning for feature extraction, it is customary to use attention-based decoding to construct an optimal solution. Thus, the operations of downstream networks rely on the quality of information extracted by GNNs. Despite their potential to learn the topology of a graph, the performance of standalone GNNs to extract important features is affected as the depth of the network increases due to over smoothing and vanishing gradient [14]. Therefore, the existing research which leverages GNNs as encoders for representation learning in solving TSP and other combinatorial problems are designed without the use of effective mechanism to control the importance of the learned features to prevent the drop of performance in GNNs as the depth increases. In addition, it is evident that the performance of GNN is highly dependent on the type of graph aggregators used to learn underlying structure of a graph [15]. Nevertheless, the existing GNN based solutions for solving TSP uses simple average pooling which does not allow graph embedding to be constructed from different representation preferences.

In this research, we contribute both in proposing novel GNN based deep reinforcement learning (DRL) model and in the experimentation. First, we propose a novel Gated Deep Graph Attention Network (G-DGANet) to efficiently learn the underlying graph topology. G-DGANet uses gating mechanism to learn important features from expanded neighborhood aggregation across different layers and designs a new graph pooling mechanism. This allows the model to aggregate rich representations of every node of a graph from its neighborhood connectivity deep into the network by allowing higher order interaction between nodes. We also encoded edge information on the link between nodes to aggregate important information of the graph during message passing. Second, during the experimentation, most of the existing research are limited to a randomly generated datasets to test the performance of a trained model, however, in this research, in addition to the test on randomly generated datasets, we also tested G-DGANet on real world road networks parsed from digital maps.

Therefore, we introduced new DRL based model with a modified GNN-based encoder to effectively learn underlying graph structure and attention-based decoder which takes the processed features from the encoder to predict the next customer node to be visited from the sequence in the graph. We trained the model with proximal policy optimization (PPO) a policy-based reinforcement learning algorithm to achieve better sample efficiency and stability during training.

The experimental results show that our proposed DRL model gives better performance than the existing learning-based approaches and heuristic methods with good generalization performance on real world instances. Generally, the contribution of our research can be summarized as follow:

1. We proposed a novel GNN based DRL framework with the introduction of gating mechanism between successive layers of GNN for solving TSP that learns features of node deeper in the graph.
2. We designed a novel gated graph pooling layer to construct global graph embedding from different representations of nodes. In addition, in view of getting better representation of graph structure, we encoded the distance information on the link between two nodes as edge features to construct rich representation of nodes.
3. In the experimentation phase, in addition to testing on random instances as existing studies, we sourced real world road network data and fit it in to the trained model for evaluation on instances obtained from real world road networks. This experiment has shown that our trained model can solve real world instances without retraining proving its generalization performance.
4. By making extensive experiments, we demonstrated that use of Gating Mechanism, gate-based graph pooling and edge encoding in the GNN can help enhance the performance of standalone GNN. To this end, G-DGANet achieved significantly better performance than the existing heuristic-based solutions and learning based solutions with better generalization from random instance training to random instance testing and real-world road network instance testing.

2. Related works

Nowadays, due to the paramount advantage of DRL as a function approximator, it is being widely used for solving COPs [16] such as TSP and vehicle routing problems (VRP) [17]. With a PtrNet [8] as the first sequence to sequence model to solve a TSP problem using a supervised learning framework, there are several other research with an increasing significance. PtrNet uses an attention mechanism to map the conditional probability of an output sequence in relation to the corresponding input sequence. This allows the output index to point to one of the input vectors at each decision step. The work in [10] presented an actor-critic reinforcement learning algorithm to address the inherent problem with the performance of the PtrNet. The proposed algorithm produced near-optimal results on TSPs with up to 100 nodes. On the other hand, [12] replaced the PtrNet based decoder by a simpler attention based RNN decoder to capture both the static and dynamic features of the VRP system. The method can also be applied to solving TSP with a comparable performance to other learning-based algorithms. In contrast to the research mentioned above, a model independent of RNN frameworks is proposed in [18]. It solely relies on attention mechanism to generate a tour for TSP with and without 2-opt procedure.

Motivated by a Transformer Network [20], a powerful method named attention model (AM) is introduced by [9] following [18] to solve VRP, Orienteering Problem, and TSP. They used REINFORCE algorithm [19] based rollout baseline to train the model. Inspired by AM, several research proposed an attention-based solutions. For instance, [20] proposed AM-RNN^2 using real time temporal traffic data for solving time dependent TSP. Encoder decoder framework is used, where the encoder additionally has an LSTM layer to process the traffic information and decoder uses GRU to generate the context embedding to be processed by the following attention layer. A related work b [13] introduced new RL model with an attention mechanism for solving COPs. Different from AM, this model applied batch normalization reordering, dynamic relational structure, and gating mechanism with an intention of enhancing context embedding. The result has shown an improvement over AM with good generalization across different distributions. Different from the above research which uses attention mechanism [21], the work in [22] proposes a novel work by adopting Talking Head attention [23] to solve TSP. In addition, the model used Gated convolutional aggregator to reduce the impact of some noise representations from attention heads. The model reportedly yielded better result and has shown good generalization performance. Some other works using DRL model for solving TSP and other combinatorial problems can also be found in [24], [25]. Our research is related to the works listed above since it makes use of an encoder-decoder framework based on reinforcement learning that leverages self-attention mechanisms in both the encoder and the decoder. However, unlike these works, we design an effective GNN based encoder for feature processing to obtain representations of a node and a graph.

GNNs are deep learning architectures which are proven to suitably encode graph structured information for better representation learning. They can effectively capture topological structures between nodes in the graph for predictions over nodes, edges, and graphs [26]. To this end, the following research can be cited as exemplary application of GNN in solving combinatorial optimization problems. Accordingly, the work of [7] demonstrated that a GNN preserves node order and performs better

in reflecting the combinatorial structure of TSP when compared to the sequence-to-sequence model. They used structure2vec (S2V) trained by a Deep Q-networks (DQN) for graph embedding. In [27], a novel neural network model is introduced which is built on an encoder-decoder architecture, with a modified graph attention network (GAT) [28] based encoder for increased node feature extraction. Inspired by [27], residual edge-based GAT (residual E-GAT) with skip connection is proposed as an encoder by [29]. They used attention-based pointer mechanism as a decoder to.

solve TSP and VRP. Their findings show that integrating edge attributes alongside node features during message aggregation enhances the learned features for graph representation learning. There are also other works with the application of graph convolution network (GCN) and Graph pointer networks [30]. For instance, the work in [31], [32] and [33] applied GCN for the embeddings of TSP graph. Different from [31] and [32], [33] used supervised learning and integrates node and edge features to obtain better topological representation of the graph. Through a comprehensive study on real-world data, it has also proved the need to explicitly encode edge features. The work in [30] proposed Graph pointer network by integrating the graph embedding to PtrNet [8]. Moreover, it also designed hierarchical graph pointer network to approximate solutions for TSP with time window which learns a reinforcement learning based policy to visit a customer node.

Inspired by the above research, we introduce a novel GNN architecture which effectively learns the representations of nodes in the structure of a graph. Compared to G-DGANet architecture, the works in [27] and [29] rely on standalone GAT, without gate mechanism between subsequent layers of the encoder. In addition, we employ an efficient graph pooling mechanism which aggregates the representation of nodes from different embeddings preferences to construct rich graph embedding in contrast to the existing research which simply takes the average of node embeddings as the graph embedding.

3. Gated deep graph attention network (G-DGANet)

3.1. Problem definition

TSP is defined as graph G with vertices V consisting of sequence of nodes $i \in V = \{x_1, x_2, \ldots, x_m\}$ where m is the size of the sequence. The features of nodes n_i is Euclidean 2D coordinate location of a node. We consider link between node i and $j(\forall\, i, j \in m)$ as the Euclidean distance between node i, and j and represent it as e_{ij}. The objective is to propose a parametrized policy that finds a permutation of nodes as a tour $\pi = (\pi_1 \ldots, \pi_m)$, which achieves minimized total travelled distance (reward), $L(\pi/s)$ given in Eq. (3.1) for visiting each node exactly once starting and ending the trip at arbitrary depot node x_{π_1}.

$$L(\pi/s) = \| x_{\pi_m} - x_{\pi_1} \|_2 + \sum_{i=1}^{m-1} \| x_{\pi_i} - x_{\pi_{i+1}} \|_2 \tag{3.1}$$

where $||\cdot||_2$ denotes the L_2 norm. G-DGANet learns a stochastic policy $p(\pi|s)$ for a given instance s to assign probability distribution to unselected nodes at every time step t to take an action π_t which is factorized and parametrized as in Eq. (3.2).

$$p_\theta(\pi_t/s) = \prod_{t=1}^{m} p_\theta(\pi_t/s, \pi_{1:t-1}) \tag{3.2}$$

We propose a new GNN-based encoder with gating mechanism which processes the informations such as positional coordinates of customer nodes n_i along with edge features e_{ij} and we use an attention-based decoder which takes the processed features from the encoder to predict the next customer node to be visited from the sequence in the graph.

3.2. Model overview

It is evident that the depth of neural networks plays a major role to learn representations of the features [34]. Similarly, the performance of GNNs heavily relies on the ability to train deep networks. Nevertheless, due to the possible reasons of over fitting, vanishing gradient and over squashing, GNNs suffer from decreasing performance as the number of layers increases, restricting the scope of information propagation among nodes. GAT [28] is one of GNN models proved to be effective for representation learning over graphs. GAT can be used for node level, edge level and graph level predictions. It is also used for representation learning of graph-structured combinatorial optimization problems like TSP [24]. However, in the attention mechanism of GAT, due to a global ranking of nodes, a node may fail to assign large edge weight to its most important neighbors restricting its expressive capability [35]. Thus, we adopted the dynamic graph attention network (GATv2)[36], a variant of GAT, to take advantage of the dynamic attention which addresses expressivity problem of original GAT. Hence, we propose a novel DRL architecture based on GATv2 to solve TSP which can better learn the node representations deeper in the network.

We used an encoder-decoder framework as shown in Fig. 1.1 and formulated the TSP task using reinforcement learning. Our encoder is made up of two distinct modules namely: Edge Embedded Gated GATv2 (E-Gated GATv2) and Graph pooling module as depicted in Fig. 3.2 These modules process the raw features to learn the underlying structure of the TSP network and provide enriched representations of each node. Generally, we make three important modifications to GATv2. The module E-Gated GATv2 embraces two of our modifications which are edge embedding and gate mechanism. First, in view of getting better representation of nodes in the graph topology, we embedded the distance information e_{ij} on the link between node i *and* j $(\forall\quad i,j \in m)$ as edge information to obtain enhanced attention weight α_{ij}^{l} at every layer $l(1, 2.. \in N)$ of G-DGANet which is based on both node n_i^l and edge representations $e_{i,j}^0$. This helps every node to propagate more information from its neighborhood connectivity to update its representation.

Secondly, inspired by the highway network [34], we introduced gating mechanism in [37] to control the importance of the learnt representations. This is basically to allow aggregation deeper in the network by preventing noise representations from propagating across the layers and to make enhanced representations of nodes. Unlike vanilla GATv2 [36] and GAT[28] which literally treats the learnt representations from different heads equally, the installed gate mechanism determines the type of information that should be passed into the next layer based on their importance. With noise representations filtered, higher order node interaction to propagate useful information is possible during computation of node embeddings. Afterwards, in the graph pooling module, we used position wise feed forward neural network (FFN) at the end of the multi head attention-based E-Gated GATv2. We get inspiration for FFN from Transformer [21].

As a final modification, we designed a gate-based graph pooling aggregator using a similar gate mechanism to effectively obtain the overall structure of a graph. To this effect, for the graph pooling operation, we first independently computed the average n_{mean} and maximum $n_{\max}$ of node embeddings and used them as input for the gate-based aggregation to obtain improved global graph embedding n_g. This type of aggregation operation assists in aggregating over various preferences to construct the graph embedding from influential node representations to obtain global features of a graph [13]. Our proposed network leverages self-attention mechanism in the encoder similar to GAT [28] and GATv2 [36]. Finally, the encoded features are passed to the attention-based decoder, which uses context embeddings calculated at each time step to predict the probabilities of unserved nodes to be visited.

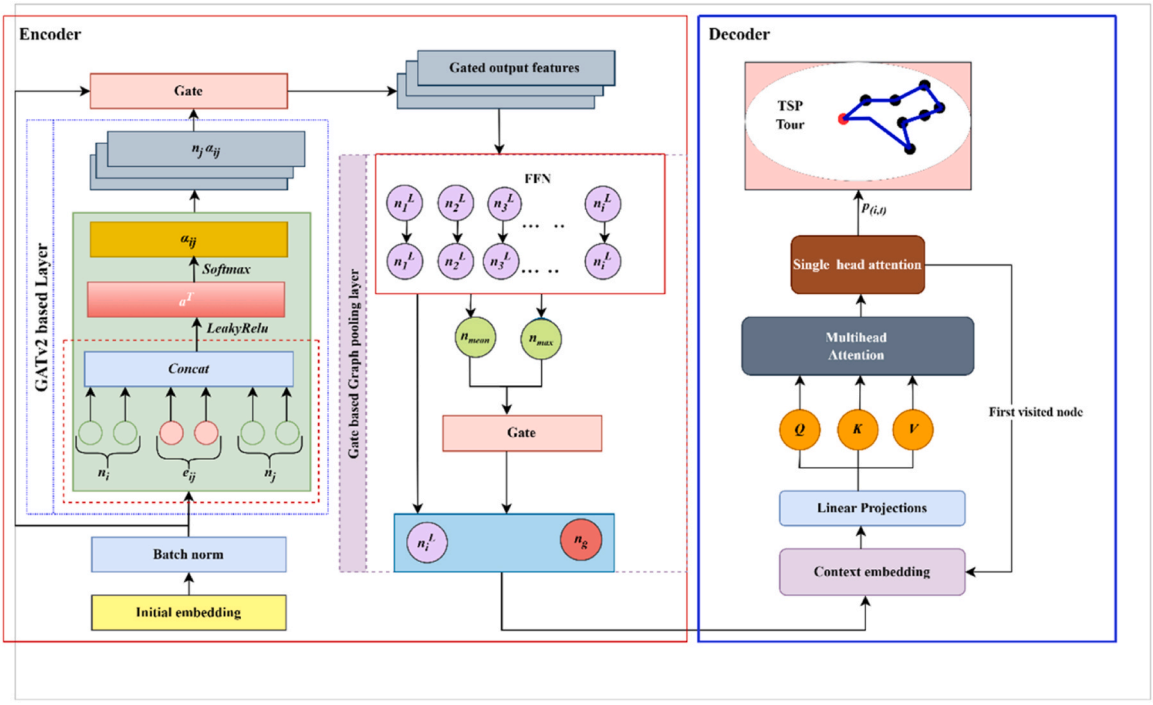


Fig. 1.1. The encoder decoder framework of G-DGANet.

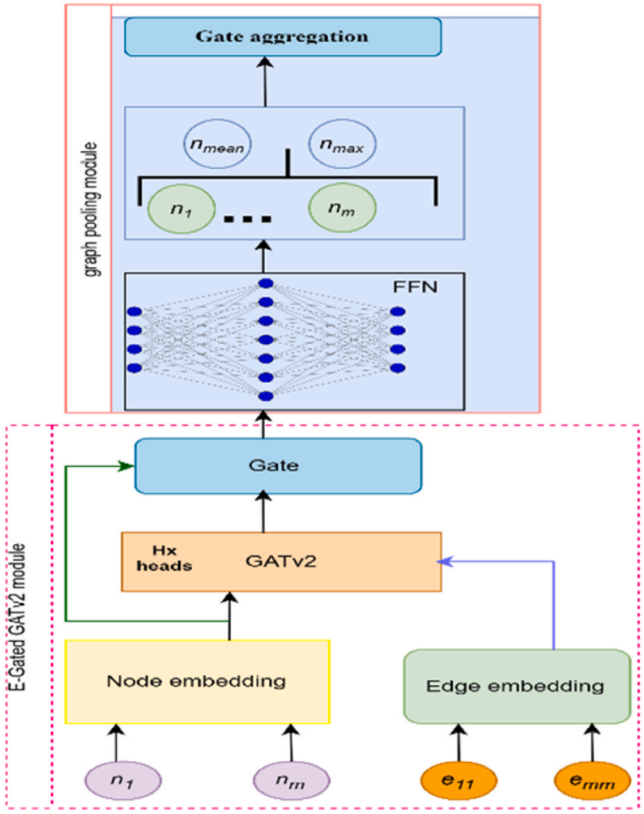


Fig. 3.2. G-DGANet encoder with E-Gated GATv2 and graph pooling module.

3.3. Encoder

As an initial operation, the input node features n_i and edge features *eij* are passed through a fully connected layer to obtain n_i^o and $e_{i,j}^0$ respectively followed by batch normalization as shown in Eqs. (3.3) and (3.4).

$$n_i^o = BN(W_n n_i + b_1) \tag{3.3}$$

$$e_{i,j}^0 = BN\left(W_e e_{ij} + b_2\right) \tag{3.4}$$

Where W_n and W_e are learnable weight matrices, BN(.) represents batch normalization. These embeddings are then fed into G-DGANet encoder for self-attention-based representation learning over graph.

3.3.1. Edge embedded gated GATv2 (E-Gated GATv2) module

Edge embedded GATv2: The initial representations of nodes n_i^o and edges $e_{i,j}^0$ from Eq. (3.3) and Eq. (3.4) are first concatenated and passed to the first layer of *E-GatedGATv2*. Since the arrays of n_i^o and $e_{i,j}^0$ are in a disparate shape, we used broadcasting for the concatenation. For all other subsequent layers, we use the embeddings of node *i* n_i^l and embeddings of node *j*, n_j^l at l^{th} layer for $l(1, 2.. \in N)$ while edge embeddings $e_{i,j}^0$ are uniform for all layers of E-Gated GATv2. Therefore, attention weight α_{ij}^l is computed as,

$$h_{ij}^{concat(l)} = \left[n_i^l \parallel n_j^l \parallel e_{i,j}^0\right] \tag{3.5}$$

$$h_{ij} = a^T LeakyRelu\left(W_l . h_{ij}^{concat}\right) \tag{3.6}$$

$$\alpha_{ij}^l = softmax_j = \frac{\exp\left(h_{i,j}\right)}{\sum_j \in N_i \exp\left(h_{i,j}\right)} \tag{3.7}$$

Where [.] is a concatenation operator, $()^T$, represents transposition, a and W_l are learnable weight matrices. The node representations are then updated by message propagation from nodes and edges based on the multi-head attention mechanism of GATv2 as shown in:

$$n_s = \sum_{j=1}^{m} \alpha_{ij}^l x_j \tag{3.8}$$

Where, W is learnable weight matrix.

Gating mechanism: We feed the embeddings of nodes from edge embedded GATv2 sublayer to the gating mechanism which takes the input and the output of the pervious layer to regulate the flow of information from one layer to succeeding layers.

$$r_t = \sigma(W_r n_s + U_r n_m) \tag{3.9}$$

$$z_t = \sigma\left(W_z n_s + U_z n_m - b_g\right) \tag{3.10}$$

$$h_t = \tanh\left(W_g n_s + U_g(r_t \Theta n_m)\right) \tag{3.11}$$

$$G(n_s, n_m) = (1 - z_t)\Theta n_s + z_t \Theta h_t \tag{3.12}$$

Where, W_r, U_r, W_z, U_z and W_g are weight matrices of linear projections, Θ denotes elementwise multiplication and n_s, n_m are the input and the outputs of l^{th} layer of the GATv2 layer respectively and, b_g= 0.1.

We perform the above operations N times to allow nodes to exchange more filtered information across different distance which improves the receptive field of every node to update its representation.

3.3.2. Graph pooling module

This module consists of two different operations which are position wise feedforward network and gate mechanism for the aggregation. **Position-wise Feedforward Networks:** The fully connected FFN is implemented with two dense linear layers with ReLU in between as shown in:

$$FF(n_i^L) = W_2 Relu(W_3 G(e_s, e_m) + b_2) + b_3) \tag{3.13}$$

Gating Mechanism: To generate the graph embedding from the enhanced representation of nodes, we first compute maximum and mean of node embeddings independently as shown in Eqs. (3.14) and (3.15) respectively to obtain different aggregation preferences.

$$n_{\max} = \max\left(n_i^L\right) \tag{3.14}$$

$$n_{mean} = \frac{\sum_{i=1}^{m} n_i^L}{m} \tag{3.15}$$

Next, we used gating mechanism to obtain the global graph embedding n_g:

$$r_t = \sigma(W_r n_{\max} + U_r n_{mean}) \tag{3.16}$$

$$z_t = \sigma\left(W_z n_{\max} + U_z n_{mean} - b_g\right) \tag{3.17}$$

$$h_t = \tanh\left(W_g n_{\max} + U_g(r_t \Theta n_{mean})\right) \tag{3.18}$$

$$n_g = g(n_{\max}, n_{mean}) = (1 - zt)\Theta \quad n_{mean} + z_t \Theta \quad h_t \tag{3.19}$$

Where, W_r, U_r, W_z, U_z, and W_g are weight matrices of linear projections, $b_g = 0.1$, σ(.) is sigmoid function.

3.4. Decoder

The decoder of our proposed approach follows the decoder design of the existing research [9], [24], [29], [38] which is based on self-attention mechanism in [21], where multi head attention (MHA) followed by a single head attention layer (SHA) is used. Simply described, the aim of decoder is to learn a route generation policy to construct a TSP route by mapping representations learnt by G-DGANet encoder into an optimal route. The decoder computes a context vector c_v^n which represents state dynamics of the system based on the initial context vector c_v^0. To get c_v^0 we concatenated the graph embedding n_g computed in Eq. (3.19), embedding of first n_1 and last selected node n_{t-1} as shown in Eq. (3.20). At time $t = 1$, we set $\overrightarrow{x}$ and $\overrightarrow{v}$, a learnable dimensional parameter d_h as a placeholder for I_x. Fig. 3.3 shows the detail decoder structure and probability distribution-based route construction.

$$c_v^o = \begin{cases} [n_g, n_1, n_{t-1}] t > 1 \\ [n_g, I_X \quad] \quad t = 1 \end{cases} \tag{3.20}$$

After computation of the context vector c_v^0, we then compute new

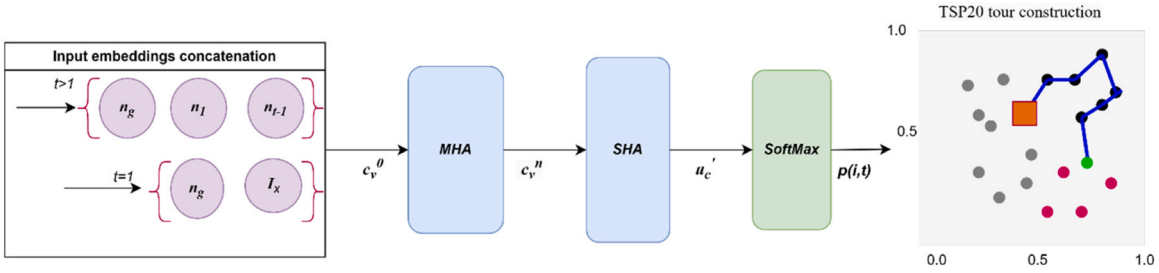


Fig. 3.3. The decoder structure and route construction.

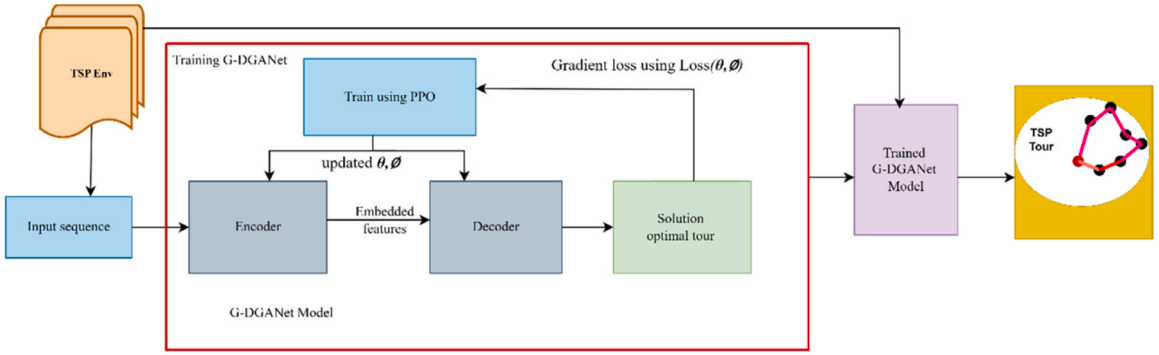


Fig. 4.4. Training and testing process of G-DGANet.

context vector denoted as c_v^n using multi-head attention mechanism [21]. We use node embeddings n_j^L to get key k_j and value vectors v_j and the context vector c_v^0 to get query vector q_j at every time step t, $t \in \{1, 2, \ldots m\}$.

$$c_v^n = MHA\left(q_j = Q_1^k c_v^0, k_j = K_1^k n_j^L, v_j = V^K n_j^L\right) \tag{3.21}$$

Where Q^k, K^k and V^k are learnable weight matrices.

Then for each node we compute attention weight from its embeddings and from the obtained context embedding.

$$q_j = Q_2^k \quad c_v^n \tag{3.22}$$

$$k_j = K_2^k \quad n_i^l \tag{3.23}$$

$$u_c^{'} = \begin{cases} C.\tanh\left(\dfrac{q_j^T \quad k_j}{\sqrt{d^k}}\right) & if \quad j \neq \pi_{i'}, \forall t^{'} \leq t \\ -\infty \text{otherwise} \end{cases} \tag{3.24}$$

Where, $Q_2^k$and K_2^k are learnable weight parameters.

The decoder observes the mask to know which customers have been visited until the current time step. By taking compatibilities as unnormalized log probabilities, for every unvisited node $i, i \in \{1, 2, \ldots m\}$ the decoder chooses the next node to be visited based on the probability $p_{i,t}$ of choosing a node at timestep t by softmax function as in:

$$p_{i,t} = p_\theta(\pi_t / s, \pi_{i'}, \forall t^{'} \leq t) = softmax\left(u_c^{'}\right) \tag{3.25}$$

Fig. 3.3 shows the detail decoder structure and probability distribution-based route construction. In the figure, the orange filled rectangle is the depot node, the line denotes the route, the dark filled dots on the line are the visited nodes, the green filled dot is the node to be visited at the current time step, the red nodes are the node with highest probability.

3.5. DRL algorithm to train our network

We adopted Proximal policy optimization [39] which is a variant of policy gradient method to train our encoder decoder-based G-DGANet. PPO is based on actor critic network and known for its better stability and sample efficiency during training from gradient based RL algorithms [40]. In our implementation, the actor network refers to our proposed model discussed in Section 3 and critic network is composed two-layered FFN, where the first layer is a dense layer with ReLU activation, and the second layer is a linear one. FFN is used as a critic to take advantage of its inherent ability to approximate complex value functions, its computational efficiency and resistance to overfitting. We used the reduced travel length $L(\pi/s)$ defined in Eq. (3.1) to obtain the unbiased Monte Carlo estimate [21] of policy gradient. Our goal is to optimize the parameters θ of the actor network that produces the best optimal policy. Advantage function A_t and the probability ratio $r(\theta)$

$$A_t = (L(\pi \backslash s) - v_\varnothing(s)) \tag{3.26}$$

$$r(\theta) = \frac{p_\theta(\pi \backslash s)}{p_{\theta old}((\pi \backslash s))} \tag{3.27}$$

Algorithm 1. PPO Algorithm for TSP.

Input: number of total epochs M, PPO steps p_s, batch size B, trainable parameters of actor network θ, behavioral actor θ_{old}, and critic network $\emptyset$.

Initialization: initialize parameter sets of θ, θ_{old} *and* φ
For each epoch =1…. M, **do**
  $s_i \sim$ Generate Random instance $s_i \forall\, i \in \{1..B\}$
  $\pi_i \sim$ sample solution $p\theta_{old}(\pi_i/s_i) \forall\, i \in \{1..B\}$
  $L(\pi_i/s_i) \sim$ Obtain reward using $\pi_i, s_i, \theta_{old}(\pi_i/s_i)$
  Put $\pi_i, s_i, \theta_{old}(\pi_i/s_i)$ and $L(\pi_i/s_s)$ in buffer.
    For each PPO_ steps $= 1, \ldots p_s$
      $v_\emptyset(s_i) \sim$ compute baseline estimate for $\forall\, i \in \{1 \ldots B\}$
      Perform reward normalization [29]
      Obtain advantage function A_t and probability ratio $r(\theta)$
      Train PPO based Actor critic network using Equation 3.26 – Equation 3.31
      Update actor and critic parameters $\theta, \emptyset$
      Update θ_{old}
    end for
end for
Output: Trained parameter set of θ

between the updated policy and the previous version of the policy are defined in Eqs. (3.26) and (3.27), respectively.

We trained the actor network using clipped surrogate objective $L_{CLIP}(\theta)$ and policy entropy loss $L_E(\theta)$ as:

$$L_{CLIP}(\theta) = E_t[\min\{ \quad r(\theta)A_t clip \quad (r(\theta), 1-\varepsilon, 1+\varepsilon)A_t \}] \tag{3.28}$$

$$L_E(\theta) = Entropy(v_\varnothing(\pi \backslash s)) \tag{3.29}$$

The critic is trained by mean squared error loss (*MSE*) that is used for updating the baseline network, determining what the discount reward will be over the run while being in state *s*.

$$L_{mse}(\varnothing) = MSE(L((\pi \backslash s), v_\varnothing(s)) \tag{3.30}$$

The total loss function of the model is then described as:

$$Loss(\varnothing, \theta) = c_p L_{clipp}(\theta) + c_v L_{mse}(\varnothing) - c_e L_E(\theta) \tag{3.31}$$

Where, c_p, c_v and c_e are the corresponding weight coefficients of, L_{clip}, L_{mse} and L_E respectively.

The detail of the adopted training is shown in Algorithm 1 briefly. The algorithm described takes PPO hyperparameters used in this research such as PPO epoch, PPO steps p_s, number of total epochs (M), the batch size (B), the trainable parameters of the actor network (θ) the

parameters of the behavioral actor (θ_{old}), and the critic network ($\varnothing$). In this research we set positive integer values 1 and 2, for PPO epoch and PPO steps p_s respectively. We also set epsilon clip ε to 0.2. We additionally use same values suggested by residual E-GAT [29] for coefficient of policy loss c_p, value loss c_v and entropy, c_e. The training and testing process is described in Figure. 4.4.

4. Experiments

In this section we report the experimental settings and experimental results obtained by implementing G-DGANet to TSP instances. We also discuss the contribution of each of our modifications discussed in Section 3 to the overall performance of the model. We implemented our proposed method using Pytorch deep learning framework and the code is written with Python 3.8. For the instance size generation and other setups, we strictly followed existing classical approaches [9], [10], [29], [41]. Accordingly, we trained G-DGANet for instance sizes of 20, 50, and 100 nodes with a batch size of 512, 128 and 128 respectively. The node coordinates are randomly generated from a uniform distribution and the demand is discrete number sampled in $\{1\ldots,9\}$. Each of our epochs uses 1000, 4000 and 4000 batches of data instance sizes of 20, 50, and 100 nodes respectively.

4.1. Hyperparameters

The node and edge embeddings dimension are 128. We also used FFN hidden dimension $d_{ff} = 512$. The encoder consists of E-Gated GATv2 module, and graph pooling module. We used 8 attention heads and 4 convolution layers for the reported result unless otherwise specified. Similarly, our decoder employs MHA with eight attention heads followed by a single-head attention layer. Following [10] the tanh clip is applied with $C = 10$. We train the models for 100 epochs on single TITAN RTX GPU. We use the Adam optimizer and set the Learning rate annealing $\eta = 10^{-4} \times 0.96^{\text{epoch}}$. 10,000 validation instances are used to evaluate the performance of G-DGANet with the selected baselines.

4.2. Comparison with Baselines

Using Gurobi [42], a commercial optimization solver that can obtain an exact solution of TSP as the benchmark, we presented our test results of the experiments in Table 4.1. Except our result and MRAM [13], the listed results are taken from [29] for greedy and sampling solutions. However, we incorporated Google OR tools [43] and Genetic algorithm (GA) by [44] as a representative solution approach of local search and evolutionary algorithm respectively to compare the performance of G-DGANet. In addition, the following learning-based baselines are used to compare G-DGANet with:

- PtrNet[8] PtrNet to generate a solution for TSP using SL.
- PtrNet[10], PtrNet with policy gradient-based RL.
- AM [9], Transformer based encoder with self-attention-based decoder with RL.
- MRAM [13], similar to AM [9] with batch normalization reordering and relational dynamic embedding.
- residual E- GAT [29], modified GAT based encoder with self-attention in decoder with RL.
- GAT [45] Attention based encoder decoder with and without 2-opt heuristics trained with RL.
- GCN [31] graph convolution network-based encoder with beam search decoder.
- S2V [46] graph embedding with q-learning.

All the results are expressed by average travel length L of test instances N_t (here $N_t = 10,000$) and optimality gap from the baseline Gurobi optimal solution.

$$Opti.Gap = \frac{1}{N_t}\sum_{i=1}^{N_t}\frac{L - L_{baseline}}{L_{baseline}} \tag{4.32}$$

As presented in Table 4.1, the proposed model outperforms all the learning-based baselines and non-learned heuristics including residual E-GAT [29] and MRAM [13] which are two strong baselines currently existing. It is worthwhile to note that G-DGANet makes use of smaller datasets when compared to the dataset requirement of the baselines. Both residual E-GAT rollout and PPO implementations are reproduced in Fig. 4.6 with a validation set of 10,000 and greedy decoding. G-DGANet converges fast with smaller amount of dataset consumption as shown in the figure for TSP20. This is because G-DGANet renders a node embedding from the encoder that can learn important features deeper in the network. This implies that the gated stacked layer of the encoder is effectively filtering influential information for downstream operations.

Table 4.1
Experimental Results (taken from Table 5.3 of [29]) H: heuristic method; SL: supervised learning; S: sample search; G: greedy search.

Method	Type	TSP20			TSP50			TSP100		
		Length	Opt.Gap	Time	Length	Opt.Gap	Time	Length	Opt.Gap	Time
Gurobi	solver	3.829	0.00%	1 m	5.698	0.00%	20 m	7.76	0.00%	3 h
Nearest insertion	H, G	4.33	12.91%	1 s	6.78	19.03%	2 s	9.46	21.82%	6 s
Random insertion	H, G	4.00	4.36%	0 s	6.13	7.65%	1 s	8.52	9.69%	3 s
Farthest insertion	H, G	3.93	2.36%	1 s	6.01	5.53%	2 s	8.35	7.59%	7 s
Nearest neighbor	H, G	4.50	17.23%	4.5 s	7.00	22.94%	0 s	9.68	24.74%	0 s
PtrNet[8]	RL, G	3.88	1.15%		7.66	34.48%				
PtrNet[10]	RL, G	3.89	1.42%		5.95	4.46%		8.30	6.90%	
S2V[7]	RL, G	3.89	1.42%		5.99	5.16%		8.31	7.03%	
GAT[18]	RL, G	3.86	0.66%	2 m	5.92	3.98%	5 m	8.42	8.41%	8 m
GAT[18]	RL, G,2opt	3.85	0.42%	4 m	5.85	2.77%	26 m	8.17	5.21%	3 h
AM[9]	RL, G	3.85	0.34%	0 s	5.80	1.76%	2 s	8.12	4.53%	6 s
MRAM[13]	RL, G	3.84	0.26%	0.37 s	5.76	1.23%	0.91 s	8.05	3.74%	2 s
GCN[31]	SL, G	3.86	0.6%	6 s	5.87	3.10%	55 s	8.41	8.38%	6 m
residual E-GAT[29]	RL, G	3.832	0.06%	1 s	5.755	0.98%	4 s	8.04	3.69%	11 s
G-DGANet	**RL, G**	**3.831**	**0.05%**	**4 s**	**5.747**	**0.85%**	**13 s**	**8.02**	**3.35%**	**30 s**
OR Tools[43]	H, S	3.85	0.37%		5.80	1.83%		7.99	2.99%	
GA[44]	H, S	3.858	0.75%	2 h	5.958	4.56%	8 h	8.327	7.26%	31 h
PtrNet[10]	RL, S				5.75	0.95%		8.00	3.03%	
GAT[17]	RL, S	3.84	0.11%	5 m	5.77	1.28%	17 m	8.75	12.70%	56 m
AM[8]	RL, S	3.84	0.08%	5 m	5.73	0.52%	24 m	7.94	2.26%	1 h
GAT[17]	RL, S,2opt	3.84	0.09%	6 m	5.75	1.00%	32 m	8.12	4.64%	5 h
residual E-GAT[29]	RL, S	3.83	0.01%	10 m	5.738	0.7%	55 m	7.896	1.71%	2 h
G-DGANet	**RL, S**	**3.83**	**0.06%**	**12 m**	**5.721**	**0.4%**	**1 h**	**7.794**	**0.43%**	**2.15 h**

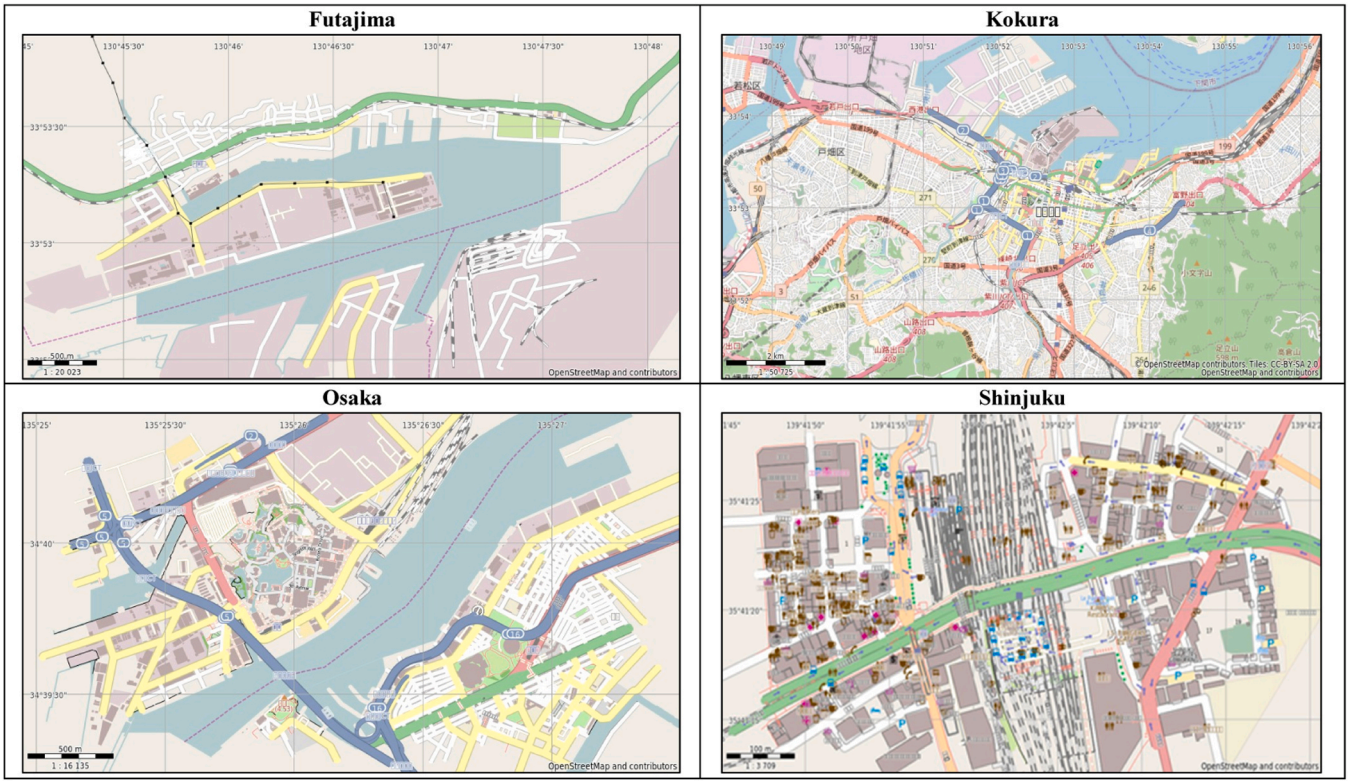


Fig. 4.5. Real world locations to extract road networks for our trained model.

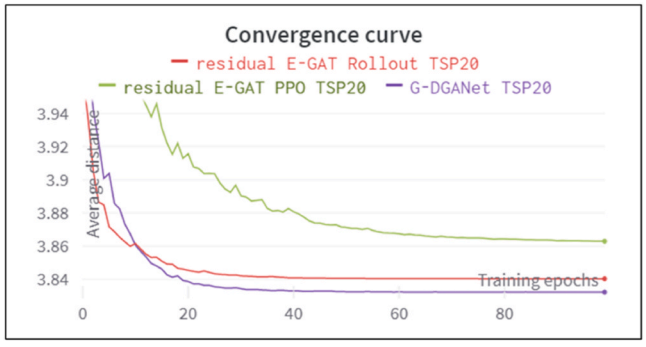


Fig. 4.6. Convergence curve of G-DGANet and baseline model.

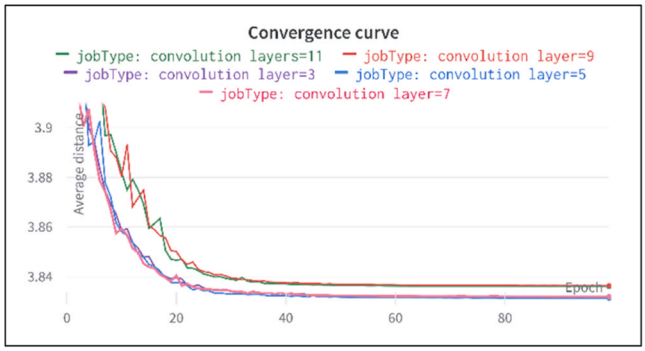


Fig. 4.7. Convergence curve of TSP20 for different number of layers.

In addition, the model also makes use of edge features which allows nodes to aggregate rich information from their neighboring nodes in the graph topology. In terms of running time complexity, G-DGANet is reasonably slower due to the additional gate operations in the encoder. However, due to the difference in the implementation language and hardware it is not worthy to make a direct comparison. Fig. 4.8 shows visualization routes of our trained model for TSP100.

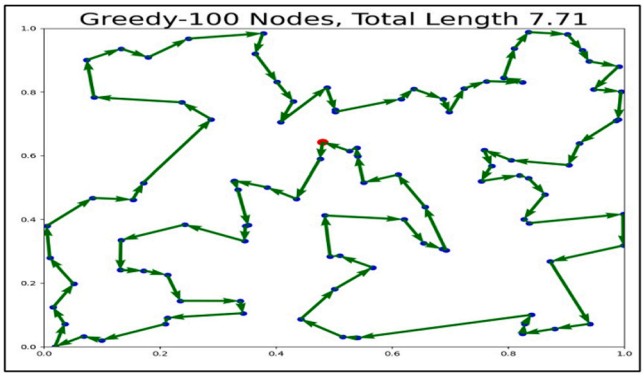


Fig. 4.8. Visualization route for TSP100.

Table 4.2
Expermental results for TSP20 with different number of layers.

Number of Convolution layers	G-DGANet TSP20		residual E-GAT TSP20	
	Length	Opt. Gap	Length	Opt. Gap
3	3.832	0.07%	3.854	0.65%
4	3.830	0.05%	3.832	0.07%
5	3.831	0.05%	3.845	0.42%
6	3.831	0.05%	3.847	0.47%
7	3.832	0.07%	3.846	0.44%
9	3.836	0.18%	3.843	0.36%
10	3.836	0.18%	3.852	0.60%
11	3.836	0.18%	3.847	0.47%

Table 4.3
Generalization for larger instance sizes.

Trained model	Generalization instance size					
	TSP20	TSP50	TSP 100	TSP 250	TSP500	TSP1000
MRAM, TSP20	3.84	5.91	8.85	15.85	24.72	37.62
AM, TSP20	3.85	5.94	9.02	16.04	24.84	38.63
G-DGANet, TSP20	**3.83**	**5.86**	**8.81**	**17.22**	**22.84**	**37.33**
MRAM, TSP50	3.87	5.76	8.12	13.77	21.25	33.03
AM, TSP50	3.87	5.79	8.14	13.95	21.61	33.66
G-DGANet, TSP50	**3.85**	**5.66**	**8.23**	**13.11**	**20.78**	**31.49**

Table 4.4
Real world test result, Kokura.

Trained model	Real world instance size			
	N = 20	N = 50	N = 100	N = 200
G-DGANet TSP20	**4.26**	**6.17**	**11.11**	**29.22**
residual E-GAT TSP20	6.36	13.76	25.91	48.72
G-DGANet TSP50	**5.10**	**6.38**	**8.60**	**11.93**
residual E-GAT TSP50	6.96	12.83	23.37	44.13
G-DGANet TSP100	**7.9**	**9.89**	**15.54**	**13.07**
residual E-GAT TSP100	8.15	11.12	16.58	26.18

4.3. Generalization of TSP models to larger instances

In addition to the results shown above, we also conducted a generalization experiment to test our trained TSP20, TSP50 and TSP100 on larger instance size. These experiments are tremendously important since it is intractable to train larger instance sizes like TSP500 and TSP1000 from scratch. For this experiment we used trained models on a different graph size with 10000 test cases for each experiment. The result.

shows that by designing a model which filters the learnt representations based on their importance it is possible to increase the generalization ability of a model which makes G-DGANet competitive against the baselines as shown in Table 4.3. Except for G-DGANet the results in the Table are all taken from Table IV of MRAM [12].

4.3.1. Sensitivity Analysis

In this section we compare the sensitivity of different number of convolution layers between G-DGANet and residual E-GAT. The number of convolution layers determines the scope of information propagation during the computation of updated node representations. Increased number of convolution layers allow higher order interaction between nodes. For instance, for convolution layer $N = 5$, the node embeddings are iteratively computed from the other node embeddings positioned as far as 5 hops of neighborhood connectivity. For this experiment we set the learning rate to $\eta = 10^{-4} \times 0.96^{\text{epoch}}$ and we conducted the experiment for TSP20. We examine the impact of applying our modifications in GATv2 module to validate its performance as the layers increase in comparison to residual E-GAT. As shown in Fig. 4.7, the performance of

Table 4.5
Real world test result, Fatajima.

Trained model	Real world instance size			
	N = 20	N = 50	N = 100	N = 200
G-DGANet TSP20	**3.98**	**4.89**	**7.01**	**14.24**
residual E-GAT TSP20	7.10	14.26	24.98	46.78
G-DGANet TSP50	**5.50**	**7.27**	**9.37**	**12.68**
residual E-GATTSP50	7.28	13.06	24.08	48.66
G-DGANet TSP100	**6.25**	**8.14**	**9.82**	**17.69**
residual E-GAT TSP100	6.45	12.56	21.18	36.43

Table 4.6
Real world test result, Osaka.

Trained model	Real world instance size			
	N = 20	N = 50	N = 100	N = 200
G-DGANet TSP20	**4.36**	**6.67**	**10.76**	**21.07**
residual E-GAT TSP20	8.89	19.79	37.07	6793
G-DGANet TSP50	**5.44**	**7.46**	**9.82**	**13.53**
residual E-GAT TSP 50	8.74	17.56	31.08	55.57
G-DGANet TSP100	**7.87**	**13.52**	**19.60**	**27.71**
residual E-GAT TSP 100	8.80	16.14	26.42	43.11

Table 4.7
Real world test result, Shinjuku.

Trained model	Real world instance size			
	N = 20	N = 50	N = 100	N = 200
G-DGANet TSP20	**4.36**	**6.47**	**10.13**	**21.71**
residual E-GAT TSP20	8.09	17.48	33.16	64.34
G-DGANet TSP50	**5.43**	**6.96**	**9.25**	**12.54**
residual E-GAT TSP 50	7.96	16.51	30.06	57.76
G-DGANet TSP100	**6.62**	**9.78**	**13.48**	**19.29**
residual E-GAT TSP 100	8.23	14.69	23.31	39.14

Table 4.8
Ablation study result.

Models	Random instances		Real-world instance (Shinjuku)
	Length	Opt. Gap	Length
G-DGANet	3.831	0.05%	4.36
G-DGANet w/o GM1	3.839	0.26%	4.39
G-DGANet w/o GM2	3.836	0.18%	4.46
G-DGANet w/o Edge	3.840	0.28%	4.38
G-DGANet w/o Feed	3.837	0.20%	3.37

G-DGANet is almost stable with competitive performance as we increase the depth of the network from 3 to 11 which is very unlikely in standalone GNNs [29], [47] [38].

Overall, it can be concluded that the use of a gating mechanism to prevent flow of noise representations across the network enabled the model to aggregate only important representations of each node from the depth of the network. Moreover, although residual E-GAT has employed residual connections to overcome the problem of vanishing gradient, as shown in Table 4.2, G-DGANet has significantly performed better. This implies that the use of gating mechanism in G-DGANet enabled the model to successfully identify important features deeper in the network by capturing the long-term dependency across layers. In addition, with the application of gating mechanism, the model tends to converge early which enhances the solution quality and the sample efficiency for all our sensitivity experiments.

4.3.2. Ablation study

To evaluate the impact of every component of our modifications, we performed an ablation study on TSP20. Specifically, we removed the key components of our modification and implemented the base model and tested it on both random instances and one selected real-world instance and presented the result as shown in Table 4.8. In addition, as depicted in Fig. 4.10 we presented the optimality gap of each epoch for each model on a validation set of 10000 with greedy decoding.

The frameworks with reduced components are named as follows:

- G-DGANet w/o GM1: G-DGANet without Gating mechanism in E-Gated GATv2

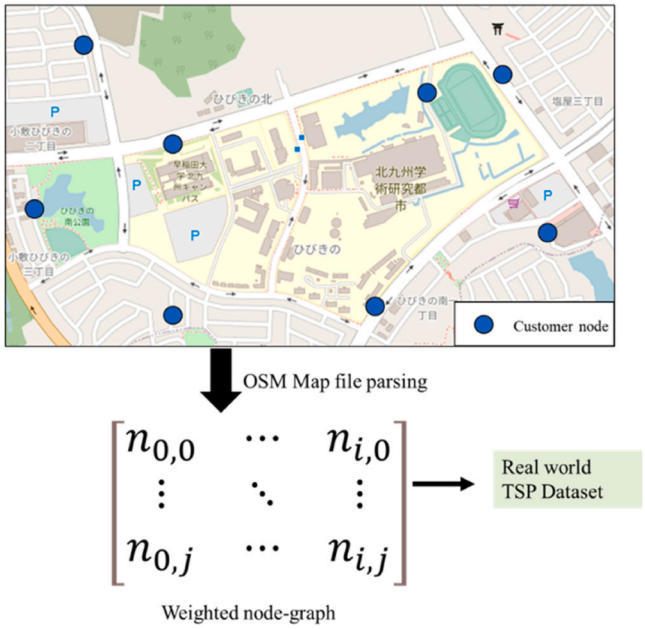


Fig. 4.9. Real world data generation process.

- G-DGANet w/o GM2: G-DGANet without Gating mechanism in graph pooling module
- G-DGANet w/o Edge: G-DGANet without edge embedding
- G-DGANet w/o Feed: G-DGANet without FFN after attention-based E-Gated GATv2

Accordingly, removing gating mechanism from G-DGANet leads into lower performance which amounts to an optimality gap of 0.39%. This implies the of importance introducing gating mechanism for the information to be filtered and distinguished before passing through successive layers.

Similarly, when removing gating mechanism in the graph pooling layer, we noticed a significant performance drop which proves the importance of using mean of node embeddings and maximum of node embeddings to aggregate into a graph embedding using gate mechanism as opposed to the simple average based aggregation in the existing studies [9], [24], [29].

By removing the edge embedding, we realized that certain amount of necessary information of the graph topology is missed, reducing the receptive field of a node in the graph. This obviously affects the aggregation of features from the neighborhood which results in lower performance.

The introduction of FFN at the end of gated multi head attention layer has demonstrated its importance which generally tells that performance of the proposed model is a combined result of all modifications

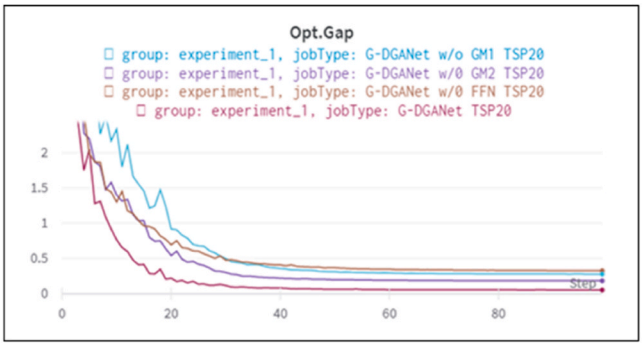


Fig. 4.10. Optimality gap of different models for ablation study.

we introduced.

4.3.3. Real-world test

One of the contributions of this research is the use of real-world road network data to evaluate the performance of the G-DGANet. For the application of our trained models on real-world graph, we implemented TSP for scenarios where the locations of customers are simulated. within a real-world graph. We used OpenStreetMap [48] as open-source repository for map data of real-world road networks. First, we sourced real-world data and transformed it into a usable format for the trained models to be tested on . We selected four locations in Japan, then implemented an algorithm that randomly selects locations for customers, with consideration to respective road network constraints in the map data. Fig. 4.5 shows the real-world locations of the case studies to extract the road network for experimentation in this study. The overall process is briefly described in Fig. 4.9.

Tables 4.4–4.7 shows the exemplary results of applying trained models on the generated datasets up to a node size of 200. Similarly, as in the previous sub-section, we focus our comparisons between our proposed model and residual E-GAT [29] (Rollout implementation) model on different instances obtained from real world road networks. We used the residual E-GAT TSP20, TSP50 and TSP100 pretrained models to generate the results from real world instances. The results shows that our trained model outperforms residual E-GAT model on most of the instances. The performance of G-DGANet is even better as the graph size increases. The reason for performance of G-DGANet is owing to the gate mechanism implemented which allows the model to distinguish important features for successive consumption in the model. Generally, the results discussed in Tables 4.4 through 4.7 show that G-DGANet can be generalized to a different and unseen type of distribution from which it is trained on. This implies that, in the proposed model some gaussian distributions are superimposed to get gaussian mixture distribution which facilitates G-DGANet to be generalized to different distributions [22]. Furthermore, the generalization performance of G-DGANet appears to be supported by the fact that GNNs learn pairwise interaction between nodes, to learn generalizable node representations, making them applicable to graph structures other than the one used for training. This attributes of GNN makes them practical to be used in the areas including robot control, wind power estimation, and city-scale weather prediction [40]. Hence, the gated stacked layers of E-Gated GATv2 played an imperative role for the generalization of G-DGANet over different instances of a real-world road network. Fig. 4.11 shows the visualizations of the routes generated by G-DGANet for Kokura test case.

5. Conclusion

In this work, we proposed an end-to-end DRL model based on novel GNN encoder to solve TSP. With the prime intention of improving node representations, we have incorporated gating mechanism in the attention-based message passing mechanism to allow filtered and influential messages pass through the successive layers of the network. To capture the graph topology and to augment the representation learnt, we also have embedded information on the link between customer nodes and employed gate-based graph pooling layer over different aggregation preferences. Our proposed model constructs a route sequentially on the basis of reducing overall distance to address every node exactly once. We adopted actor critic-based PPO for training the model. Our experiments showed that our proposed model has better performance than heuristic solvers and other learning based existing baselines. Experiments on real world road networks similarly showed that our proposed model is generalizable from random instance training to real world instance testing. As future work, we consider expanding this work to address other variants of the routing problems and combinatorial optimization problems with various experiments in different decoding mechanism such as active search and beam search approaches. More

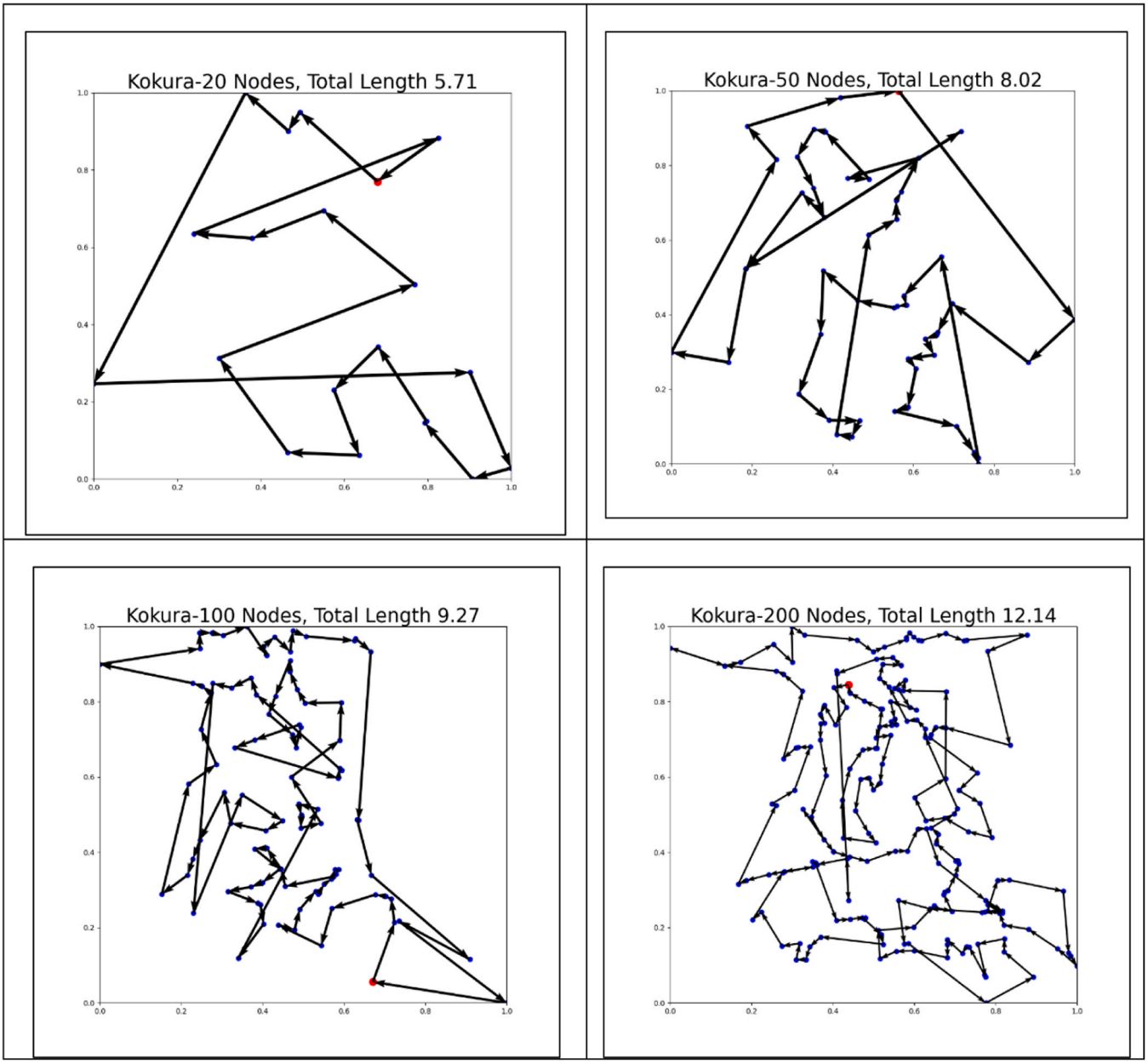


Fig. 4.11. Exemplary Routes on real world graph Kokura test case (red and blue dots represent the depot and the customer nodes on the road network respectively.

importantly with the consideration of temporal information on the road network between customer nodes, flexible models can be built to address problems of practical importance. As for the utilization of real-world road network datasets, we believe that this work have made the first step in using such data to test the trained models. Moreover, the utilization of such datasets for the training and testing of neural networks can also be the focus of future research; for instance, studying deployment approaches that ensure scalability on larger node graphs in practical settings.

Funding

FELLEK GETU TADESSE reports financial support was provided by Japan International Cooperation Agency.

CRediT authorship contribution statement

Getu Fellek: Conceptualization, Methodology design and development, Software, Experimental analysis, Validation, Writing -Original draft. **Ahmed Farid**: Conceptualization of real-world test, Software, Review. **Shigeru Fujimura**: Writing, Review and editing, Supervision, Project administration. **Osamu Yoshie**: Writing, Review and editing, Supervision, Project administration. **Goytom Gebreyesus**: Review and editing.

Declaration of Competing Interest

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

Data availability

Data will be made available on request.

References

[1] M. Chen, L. Gao, Q. Chen, Z. Liu, Dynamic Partial Removal: A Neural Network Heuristic for Large Neighborhood Search, May 2020, [Online]. Available: ⟨http://arxiv.org/abs/2005.09330⟩.

[2] M.H. Miraz, Institute of Electrical and Electronics Engineers, and Institute of Electrical and Electronics Engineers. United Kingdom and Republic of Ireland Section, 2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE): proceedings: University of Essex, Southend, UK, 16th-17th August, 2018.

[3] I. Drori *et al.*, "Learning to Solve Combinatorial Optimization Problems on Real-World Graphs in Linear Time," in 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, Dec. 2020, pp. 19–24. doi: 10.1109/ICMLA51294.2020.00013.

[4] Y. Kaempfer and L. Wolf, "Learning the Multiple Traveling Salesmen Problem with Permutation Invariant Pooling Networks," Mar. 2018, [Online]. Available: ⟨http://arxiv.org/abs/1803.09621⟩.

[5] N. Bianchessi, G. Righini, Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery, Comput. Oper. Res vol. 34 (2) (2007) 578–594, https://doi.org/10.1016/j.cor.2005.03.014.

[6] S.O. Krumke, S. Saliba, T. Vredeveld, S. Westphal, Approximation algorithms for a vehicle routing problem, Math. Methods Oper. Res. vol. 68 (2) (2008) 333–359, https://doi.org/10.1007/s00186-008-0224-y.

[7] Elias B. Hanjun Dai, Yuyu Khalil, Bistra Zhang, Dilkina, and le Song, "learning combinatorial optimization algorithms over graphs,", 31st Conf. Neural Inf. Process. Syst. (NIPS 2017 (2017) 1–11.

[8] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, Proc. Adv. Neural Inf. Process. Syst. (NeurIPS) (2015) 2692–2700.

[9] W. Kool, van Hoof, M. Welling, Attention, learn to solve routing problems, Proc. Int. Conf. Learn. Represent. (ICLR) (2019) 1–25.

[10] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, Int. Conf. Learn. Represent. (2017).

[11] G. Fellek, A. Farid, G. Gebreyesus, S. Fujimura, O. Yoshie, Graph transformer with reinforcement learning for vehicle routing problem, IEEJ Trans. Electr. Electron. Eng. (2023), https://doi.org/10.1002/tee.23771.

[12] Mohammadreza Nazari, Afshin Oroojlooy, V.Snyder Lawrence, Martin Taká̆, Neural combinatorial optimization with reinforcement learning, Proc. 32nd Int. Conf. Neural Inf. Process. Syst. 2018 (2018) 9861–9871.

[13] Y. Xu, M. Fang, L. Chen, G. Xu, Y. Du, C. Zhang, Reinforcement learning with multiple relational attention for solving vehicle routing problems, IEEE Trans. Cyber (2021), https://doi.org/10.1109/TCYB.2021.3089179.

[14] L. Zhao and L. Akoglu, "PairNorm: Tackling Oversmoothing in GNNs," in *ICLR*, Sep. 2020.

[15] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs," Mar. 2018, [Online]. Available: ⟨http://arxiv.org/abs/1803.07294⟩.

[16] J. Li, X. Dong, K. Zhang, S. Han, "Solving Open Shop Scheduling Problem via Graph Attention Neural Network," in Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI, IEEE Computer Society, Nov. 2020, pp. 277–284. doi: 10.1109/ICTAI50040.2020.00052.

[17] R. Zhang, A. Prokhorchuk, J. Dauwels, "Deep Reinforcement Learning for Traveling Salesman Problem with Time Windows and Rejections," in 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, Jul. 2020, pp. 1–8. doi: 10.1109/IJCNN48605.2020.9207026.

[18] Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.-M., Learning Heuristics for the TSP by Policy Gradient, 2018, pp. 170–181. doi: 10.1007/978-3-319-93031-2_12.

[19] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn vol. 8 (3–4) (1992) 229–256, https://doi.org/10.1007/BF00992696.

[20] G. Wu, Z. Zhang, H. Liu, J. Wang, Solving Time-Dependent Traveling Salesman Problem with Time Windows with Deep Reinforcement Learning. Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics, Institute of Electrical and Electronics Engineers Inc.,, 2021, pp. 558–563, https://doi.org/10.1109/SMC52423.2021.9658956.

[21] Ashish Vaswani, et al., Attention is all you need, *Adv. Neural Inf. Process. Syst. 30 (NIPS 2017)* (2017) 5998–6008.

[22] S. Guo, Y. Xiao, and L. Niu, "GGTAN: Graph gated talking-heads attention networks for traveling salesman problem," in Proceedings - 2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2020, Institute of Electrical and Electronics Engineers Inc., Dec. 2020, pp. 676–681. doi: 10.1109/WIIAT50758.2020.00102.

[23] N. Shazeer, Z. Lan, Y. Cheng, N. Ding, and L. Hou, "Talking-Heads Attention," Mar. 2020, [Online]. Available: ⟨http://arxiv.org/abs/2003.02436⟩.

[24] X. Bresson and T. Laurent, "The Transformer Network for the Traveling Salesman Problem," Mar. 2021, [Online]. Available: ⟨http://arxiv.org/abs/2103.03012⟩.

[25] IEEE Computational Intelligence Society, International Neural Network Society, Institute of Electrical and Electronics Engineers, and IEEE World Congress on Computational Intelligence (2020: Online), 2020 International Joint Conference on Neural Networks (IJCNN): 2020 conference proceedings.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P.S. Yu, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn Syst. vol. 32 (1) (2021) 4–24, https://doi.org/10.1109/TNNLS.2020.2978386.

[27] L. Gao, M. Chen, Q. Chen, G. Luo, N. Zhu, and Z. Liu, "Learn to Design the Heuristics for Vehicle Routing Problem," Feb. 2020.

[28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph Attention Networks, Int. Conf. Learn. Represent., 2017 (2017).

[29] K. Lei, P. Guo, Y. Wang, X. Wu, W. Zhao, Solve routing problems with a residual edge-graph attention neural network, Neurocomputing vol. 508 (2022) 79–98, https://doi.org/10.1016/j.neucom.2022.08.005.

[30] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, "Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning." [Online]. Available: ⟨www.aaai.org⟩.

[31] C.K. Joshi, T. Laurent, and X. Bresson, "An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem," Jun. 2019, [Online]. Available: ⟨http://arxiv.org/abs/1906.01227⟩.

[32] Z. Li, Q. Chen, and V. Koltun, "Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search."

[33] L. Duan, et al., Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, 2020, pp. 3054–3063, https://doi.org/10.1145/3394486.3403356.

[34] R.K. Srivastava, K. Greff, J. Schmidhuber, "Highway Networks," May 2015, [Online]. Available: ⟨http://arxiv.org/abs/1505.00387⟩.

[35] K. Fountoulakis, A. Levi, S. Yang, A. Baranwal, A. Jagannath, "Graph Attention Retrospective," Feb. 2022, [Online]. Available: ⟨http://arxiv.org/abs/2202.13060⟩.

[36] S. Brody, U. Alon, and E. Yahav, "How Attentive are Graph Attention Networks?," May 2021, [Online]. Available: ⟨http://arxiv.org/abs/2105.14491⟩.

[37] E. Parisotto *et al.*, "Stabilizing Transformers for Reinforcement Learning," 2020. [Online]. Available: ⟨www.github.com/tensorflow/tensor2tensor⟩.

[38] K. Zhang, F. He, Z. Zhang, X. Lin, M. Li, "Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach, Transp. Res Part C. Emerg. Technol. vol. 121 (2020), https://doi.org/10.1016/j.trc.2020.102861.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Jul. 2017, [Online]. Available: ⟨http://arxiv.org/abs/1707.06347⟩.

[40] J. Park, J. Chun, S.H. Kim, Y. Kim, J. Park, "Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning,", Int J. Prod. Res vol. 59 (11) (2021) 3360–3377, https://doi.org/10.1080/00207543.2020.1870013.

[41] G. Fellek, G. Gebreyesus, A. Farid, S. Fujimura, and O. Yoshie, "Edge Encoded Attention Mechanism to Solve Capacitated Vehicle Routing Problem with Reinforcement Learning," in 2022 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), IEEE, Dec. 2022, pp. 576–582. doi: 10.1109/IEEM55944.2022.9989997.

[42] "Gurobi optimization software."

[43] Google, "Google OR Tools," 2022. ⟨https://developers.google.com/optimization⟩ (accessed Feb. 25, 2022).

[44] L. Jacobson and B. Kanber, *Genetic algorithms in Java basics*.

[45] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning Heuristics for the TSP by Policy Gradient."

[46] H. Dai, E.B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning Combinatorial Optimization Algorithms over Graphs," Apr. 2017, [Online]. Available: ⟨http://arxiv.org/abs/1704.01665⟩.

[47] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE,, 2016, pp. 770–778, https://doi.org/10.1109/CVPR.2016.90.

[48] M. Haklay, P. Weber, OpenStreetMap: user-generated street maps, IEEE Pervasive Comput. vol. 7 (4) (2008) 12–18, https://doi.org/10.1109/MPRV.2008.80.

Getu Fellek Received B.Sc. Degree and M.Sc. Degree from Mekelle University, Ethiopia in 2014 and 2016 respectively. He is currently pursuing Ph.D. at Graduate School of Information, Production and Systems, Waseda University Major in Informatics. His research interests include machine learning based automation, reinforcement learning and routing and navigation system.

Ahmed Farid Received the B.Sc. Degree from October University for Modern Sciences & Arts, Egypt, in 2013, and the M. Sc. Degree from Waseda University, Japan, in 2018. He is currently pursuing Ph.D. at Graduate School of Information, Production and Systems, Waseda University. His research interests include computer vision, autonomous driving, and data engineering.

Shigeru Fujimura Received B.E., M.E., and Dr. Eng. degrees from Waseda University in 1983, 1985, and 1995, respectively. He joined Yokogawa Electric Corporation in 1985 and worked there until 2003. Since 2003, he has been a Professor at Graduate School of Information, Production and Systems, Waseda University. His research interests include production management, production planning and scheduling, intelligent agents, object-oriented modeling, and business process modeling.

Osamu Yoshie Born in 1959 and obtained a doctor of engineering degree from Waseda University, he is currently a professor at Graduate School of Information, Production and Systems, Waseda University and Director of Institute for Global Strategies on Industry-Academia Fusion. His research interests include community-oriented information processing (especially consensus building process analysis) and industrial applications of machine learning. He has received the Society of Instrument and Control Engineers (SICE) Award for Academic Encouragement, the Institute of Electrical Engineers of Japan (IEEJ) Award for Outstanding Paper for the Promotion of Science, the Japan Society of Facilities Management, the Japan Society of Management Engineers (JSME) Award for Management Systems, the IEEJ Achievement Award, and others. He is a Fellow of the IEEJ.

Received the B.Sc. Degree from Mekelle University, Ethiopia, in 2010, and M.Sc. Degree from Addis Ababa University, Ethiopia, in 2013. He is currently pursuing Ph.D.at Graduate School of Information, Production and Systems, Waseda University. His research interests include deep reinforcement learning based industrial systems automation, production planning and scheduling using machine learning and deep learning, data engineering, and process optimization.