Graph Ordering: Towards the Optimal by Learning

Kangfei Zhao
kfzhao@se.cuhk.edu.hk
The Chinese University of Hong Kong

Yu Rong
yu.rong@hotmail.com
Tencent AI Lab

Jeffrey Xu Yu
yu@se.cuhk.edu.hk
The Chinese University of Hong Kong

Junzhou Huang
joehhuang@tencent.com
Tencent AI Lab

Hao Zhang
hzhang@se.cuhk.edu.hk
The Chinese University of Hong Kong

ABSTRACT
Graph representation learning has achieved a remarkable success in many graph-based applications, such as node classification, link prediction, and community detection. These models are usually designed to preserve the vertex information at different granularity and reduce the problems in discrete space to some machine learning tasks in continuous space. However, regardless of the fruitful progress, for some kind of graph applications, such as graph compression and edge partition, it is very hard to reduce them to some graph representation learning tasks. Moreover, these problems are closely related to reformulating a global layout for a specific graph, which is an important NP-hard combinatorial optimization problem: graph ordering. In this paper, we propose to attack the graph ordering problem behind such applications by a novel learning approach. Distinguished from greedy algorithms based on pre-defined heuristics, we propose a neural network model: Deep Order Network (DON) to capture the hidden locality structure from partial vertex order sets. Supervised by sampled partial order, DON has the ability to infer unseen combinations. Furthermore, to alleviate the combinatorial explosion in the training space of DON and make the efficient partial vertex order sampling, we employ a reinforcement learning model: the Policy Network, to adjust the partial order sampling probabilities during the training phase of DON automatically. To this end, the Policy Network can improve the training efficiency and guide DON to evolve towards a more effective model automatically. The training of two networks is performed interactively and the whole framework is called DON-RL. Comprehensive experiments on both synthetic and real data validate that DON-RL outperforms the current state-of-the-art heuristic algorithm consistently. Two case studies on graph compression and edge partitioning demonstrate the potential power of DON-RL in real applications.

1 INTRODUCTION
Graphs have been widely used to model complex relations among data, and have been used to support many large real applications including social networks, biological networks, citation networks and road networks. In the literature, in addition to the graph algorithms and graph processing systems designed and developed, graph representation learning have also been studied recently [3, 7, 14, 15, 19, 21, 29, 31, 35, 37, 41, 46]. They are designed to preserve vertex information at different granularity from microscopic neighborhood structure to macroscopic community structure, and are effectively used for node classification, link prediction, influence diffusion prediction, anomaly detection, network alignment, and recommendation. The surveys can be found in [12, 20].

In this paper, we concentrate ourselves on a rather different set of graph problems: graph visualization [6], graph compression [22], graph edge partitioning [9, 45]. Here, graph visualization is to provide an effective way to visualize complex data, graph compression is to find a compact edge layout, and graph edge partitioning is to partition edges instead of vertices for better load balancing [16, 17]. Such problems can be addressed if we can arrange all vertices in a good order. The state-of-art graph ordering [42] reassigns every vertex a unique number in [1..n] where n is the number of vertices in the graph by maximizing a cumulated locality score function for vertices in a sliding window of size w (>1), which is an NP-hard combinatorial optimization problem as a general form of the maximum Travel Salesman Problem (TSP) (w=1). The GO algorithm...
proposed in [42] achieves $\frac{1}{2K}$-approximation, and its real performance is very close to the optimal in their testing, as it is close to the upper bound of the optimal solution.

Given such a high-quality solution for graph ordering, a natural question is if we can do better by learning. The answer is yes. We demonstrate it in Fig. 1, which shows the matrix visualization for the Facebook [27] reordered by the GO algorithm and our new learning-based approach over graph. The left figures are the overall visualizations and the right are the zoom-in details of the upper-left part of the overall ordering. Comparing with the GO algorithm [42] (Fig. 1(a)), our learning-based approach (Fig. 1(b)) keeps the compact permutation better from the local perspective, and has the advantage of avoiding stucking in forming local dense areas from the global perspective. Moreover, the learned permutation has a large potential to support other real applications. Based on our case studies, the learned permutation reduces the storage costs larger than 2X compared with the order for real-graph compression [22]. And the edge partitioning deriving from our approach outperforms the widely-used partitioning algorithm [9] up to 37%.

The main contributions of this paper are summarized below.

- For the graph ordering problem, distinguished from the traditional heuristics-based algorithm, we propose a neural network-based model: Deep Order Network (DON) to replace the predefined evaluation function to capture the complex implicit locality in graphs. Hence our proposed model can produce a better graph ordering than the current state-of-the-art heuristics-based algorithm.

- To address the combinatorial explosion in the training space of DON, we exploit the idea of reinforcement learning and construct a policy network to learn the sampling probability of vertices. This policy network enables DON to focus on vertices which have more remarkable contribution in the solution and guides DON to evolve towards a more effective model automatically. The whole learning framework is called DON-RL.

- We conduct extensive experiments on both synthetic and real-world graphs with different properties. Experimental results show that DON-RL consistently outperforms the other algorithms. Furthermore, we conduct two case studies: graph compression and graph edge partitioning to demonstrate the potential value of our algorithm in real applications.

The paper is organized as follows. Section 2 introduces the preliminaries, including the problem definition, the intuitions and challenges. Section 3 gives an overview of our learning framework, which is composed of a central network for ordering and an auxiliary Policy Network. The design principles of these two neural networks are introduced in Section 4 and Section 5, respectively. Section 6 presents the experimental result of our approach in solving the graph ordering problem. We review the related works in Section 7. Finally, we conclude our approach in Section 8.

2 PRELIMINARIES

In this section, first, the graph ordering problem and its existing solution are introduced. Then, we elaborate on our intuitive improvement and the challenges.

Table 1: Frequently Used Notations

| Notations | Definitions |
|-----------|-------------|
| $G = (V, E)$ | directed graph with the vertex set $V$ and the edge set $E$. |
| $S(u, v)$ | The pair-wise similarity function of vertex $u$ and $v$. |
| $F(\cdot)$ | The locality score function of a permutation $\cdot$. |
| $w$ | The given window size. |
| $Q(S, v; \Theta)$ | Deep Order Network (DON). |
| $D(\cdot)$ | Policy Network. |
| prob (grapht) | The vertices sampling probability (at time $t$). |
| $\hat{S}(\cdot)$ | The vertices sampling probability (at time $t$). |
| $D_t$ | The training data at time $t$. |
| $\mathbf{D}_t$ | The evaluation data. |

Algorithm 1 GO ($G = (V, E)$)

1. $S \leftarrow \emptyset$;
2. while $|S| \neq |V|$ do
3. \hspace{1em} $v^* = \max_{v \in V \setminus S} \Phi(S, v)$;
4. \hspace{1em} $S \leftarrow S \cup \{v^*\}$;
5. end while
6. return $S$;

Figure 2: An Example of Graph Ordering [42]

2.1 Graph Ordering

Given a graph $G = (V, E)$ where $V(G)$ is a set of vertices and $E(G)$ is a set of edges of $G$, the graph ordering problem is to find the optimal order for all vertices. In brief, let $\Phi(\cdot)$ be a permutation function which assigns a vertex a unique number in $[1..n]$ where $n = |V(G)|$. The problem is to find the optimal graph ordering $\Phi(\cdot)$ by maximizing a cumulated locality score $F(\cdot)$ (Eq. (1)) over a sliding window of size $w$.

$$F(\Phi) = \sum_{0 \leq \Phi(v) - \Phi(u) \leq w} S(u, v) \quad (1)$$

Here, $u, v$ are two vertices ordered within a window of size $w$ in the permutation, and $S(u, v)$ is a pair-wise similarity function to measure the closeness of $u$ and $v$. In [42], $S(u, v)$ is defined as $S(u, v) = S_1(u, v) + S_2(u, v)$, where $S_1(u, v)$ is the number of times that $u$ and $v$ are sibling, and $S_2(u, v)$ is the number of times that $u$ and $v$ are neighbors.

Example 2.1: Taking the directed graph shown in Fig. 2(a) as an example, there are 12 nodes numbered from 1 to 12, which is considered as one possible graph ordering. Table 2 lists the pair-wise $S(u, v)$ values, in a matrix form for vertices 1 – 5 in Fig. 2(a) due to limited space. Given a window size $w = 3$, consider a partial permutation of length 3, $\Phi = [1, 2, 5]$ for the vertices in Fig. 2(a), its locality score $F(\Phi) = S(1, 2) + S(2, 5) + S(3, 5) = 4$. Fig. 2(b) shows the optimal graph ordering which maximizes $F(\Phi)$ given the window size $w = 3$. \qed
In general, maximizing $F(\Phi)$ is a variant of maximum TSP (for $w = 1$) with the sum of weights within a window of size $w > 1$ and thus is NP-hard as proven in [42].

Wei et al. in [42] proposed a greedy algorithm (named $GO$), as sketched in Algorithm 1, which iteratively extends a partial solution $S$ by a vertex $v^*$. The vertex $v^*$ is selected by maximizing a potential function $Q(S, v)$, which measures the quality of the partial solution extended by a vertex $v$, based on the current partial solution $S$.

In details, in $i$-th iteration, it computes the cumulated weight $k(v)$ (Eq. 2) for each vertex $v$ and inserts the one with maximum $k(v)$ in the permutation:

$$k(v) = \sum_{j=\max(1, i-w)}^{\max(1, i-w)} S(v_j, v)$$

where $k(v)$ serves as the potential function $Q$. $\{v_j\}_{j=\max(1, i-w)}$ are the last $\max(1, i-w)$ inserted vertices. This procedure repeats until all the vertices in $G$ are placed. The total time complexity of $GO$ is $O(d_{\text{max}}n^2)$, where $d_{\text{max}}$ denotes the maximum in-degree of the graph $G$. There are $n$ iterations, and in each iteration, it scans the remaining vertices in $O(n)$ and computes the score $k(v)$ for the scanned vertex $v$ in $O(d_{\text{max}})$. The algorithm achieves $\frac{1}{2w}$ approximation for maximizing $F(\Phi)$. It can achieve high-quality approximations with small window size and its practical performance is close to the optimal in the experiments.

Table 2: $S(u, v)$ Values for Fig. 2(a)

| $S(u, v)$ | 1 | 2 | 3 | 4 | 5 |
|-----------|---|---|---|---|---|
| 1         | 1 | 0 | 0 | 0 | 0 |
| 2         | 1 | 1 | 0 | 0 | 0 |
| 3         | 1 | 1 | 1 | 0 | 0 |
| 4         | 1 | 1 | 1 | 0 | 0 |
| 5         | 1 | 1 | 1 | 1 | 0 |

for all states recursively:

$$Q(s, a) = \mathbb{E}[r(s, a) + \gamma \max_{a'} Q(s', a')|s, a],$$

where $Q(s, a)$ and $Q(s', a')$ are the maximum expected cumulative rewards achievable from the state-action pair $(s, a)$ and the previous state-action pair $(s', a')$, respectively. To avoid exhaustively exploring the MDP, Monte Carlo simulation is used to sample the state-action trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_T, a_T, r_T)$ to estimate the values of downstream state-action pairs. In this vein, DRL is adopted to solve the combinatorial optimization problem [8, 23], whose policy is learned by specific deep neural networks.

Challenges: Recently, these RL-based approaches [8, 23] have surpassed traditional greedy strategies on finding better feasible solutions. However, in practice, the pure RL framework is inefficient as the discrete action space becomes large. This drawbacks obstacles RL-base approaches to solve the graph ordering problem over a single large real-world graph. Concretely, RL algorithms use Thompson Sampling [18] to explore trajectories and estimate the expected cumulative reward (Eq. (3)). The larger the action space is, the lower probability trajectories with high accumulated reward can be sampled in limited steps. The low-quality samples will extremely degrade the effectiveness of the model as well as the convergence in the training phase. Therefore, [8, 23] can only support training on graphs with tens to one hundred of vertices, where their optimal solutions are provided by state-of-the-art integer programming solvers [5, 10]. Recall that AlphaGo [32] has $19 \times 19$ actions at most, which in fact is an enormously large action space requiring high computation power of a large cluster to explore. Unfortunately, due to the combinatorial explosion in the training space of $Q(S, v)$, any real large graph has a larger searching space than that of AlphaGo.

In a nutshell, the construction of the parametric model $Q(S, v; \Theta)$ and the efficient training in an exponential training space are two main challenges when we try to adopt the model-based framework to solve the graph ordering problem.

2.2 Intuitions and Challenges

Intuitions: Given the current greedy-based framework, Algorithm 1 can be regarded as a discrete Markov Decision Process (MDP) [34], where the states are feasible partial permutations, actions are to insert one vertex to the permutations. For traditional greedy algorithm with predefined heuristics, the solution is built by a deterministic trajectory which has locally optimal reward. With regards to the graph ordering problem, $Q(s, a)$ (Eq. (3)) is simply approximated by the evaluation function $Q(S, v)$, which is specifically defined in Eq. (2). A natural question arises that can we do better by learning an expressive, parameterized $\hat{Q}(S, v; \Theta)$ instead.

Inspired by universal approximation theorem [11], deep reinforcement learning (DRL) learns a policy $\pi$ from states to actions that specifies what action to take in each state. We can formulate Algorithm 1 as MDP, which is composed by a tuple $(S, A, R, P, \gamma)$:

1. $S$ is a set of possible states; 2. $A$ is a set of possible actions; 3. $R = r(s, a_t)$ is the distribution of reward given state-action $(s_t, a_t)$ pair; 4. $P = P(s_{t+1}|s_t, a_t)$ is the transition probability given $(s_t, a_t)$ pair; 5. $\gamma$ is the reward discount factor. The policy $\pi$ is a function $\pi: S \rightarrow A$ that specifies what action to perform in each state.

The objective of RL is to find the optimal policy $\pi^*$ which maximizes the expected cumulative discounted reward. This cumulative discounted reward can be defined by the Bellman equation as below.

3 A NEW APPROACH: DON-RL

In order to address the challenges mentioned in Section 2.2, we propose a new framework: Deep Ordering Network with Reinforcement Learning (DON-RL) to capture the complex combinatorial structures of a single large graph in solving the graph ordering problem. To make the better approximation of the evaluation function $Q(S, v)$, we propose a new model: Deep Order Network (DON) which takes the partial vertex order set $S$ as input and predict the next vertex which can maximize the cumulated locality score. Furthermore, to prune the exponential training space and improve the training of DON efficiency, we employ a policy gradient RL model; the Policy Network to explore the valuable vertex order segments efficiently. Fig. 3 demonstrates the overall learning framework.

Deep Order Network: As a parametric permutation-invariant model of the evaluation function $Q(S, v)$ (Algorithm 1), DON takes a partial vertex order solution, i.e., a set of vertices as input, and outputs the probability distribution of the decision of next vertex to be selected. In other words, DON is to learn a parameterized $\hat{Q}(S, v; \Theta)$ function by exploiting vertex sets over the whole graph $G$, where every set encodes the contribution of vertices in the set for constructing $F(\Phi)$ if they coexist in a window of size $w$. DON
is trained by supervised learning over partial solutions. Here, a partial solution is a set of \( m \leq w \) vertices for a window of size \( m \) by first sampling \( m - 1 \) vertices from \( G \) followed by computing the likelihood of the last vertex to be in the window. The last vertex to be added for a partial solution is by the ground truth of the \( Q \) value, which is easy to be calculated when \( m \) is small.

Since the function \( \hat{Q}(S, v; \Theta) \) has the property of permutation invariant regarding any permutation of the vertices in \( S \), we employ a permutation invariant neural network architecture to model it. We present the design and principle of our Deep Order Network in Section 4 in details.

**RL-based Training:** In practice, the space of the all possible partial vertex order solution is exponential with respect to the number of vertices of a graph. Theoretically, a neural network can memorize and serve as an oracle for any query solution. However, it is intractable to enumerate all the partial solutions in training DON. Hence, sampling the high-quality partial solutions, i.e., sets of vertices, is vital to train DON model with high generalization ability which can inference unseen vertex combinations rather than memorization.

In other words, DON should have the ability to inference unseen vertex combinations rather than memorization. Hence, sampling the high-quality partial solutions, i.e., sets of vertices, is vital for the training of DON.

Rather than specifying the sampling probability uniformly or by human experience, we employ Policy Network to explore how to make adjustments on the probability, based on the evaluation result of DON dynamically. In this vein, we regard DON as an environment and the Policy Network is an agent which obtains rewards from the environment and predicts an adjusting action. The sampler of vertex sets adopts the adjusted sampling probability (i.e., the state transformed from adjusting action of the Policy Network), to sample the training data for next time slot and feeds these data to DON. After DON is trained for a certain number of iterations, the evaluation result of DON is received by the Policy Network as the rewards for updating its weights, as well as subsequent prediction. The training of the two networks is performed interactively, where the updating of the Policy Network is controlled by the evaluation result of DON.

The significance of using an RL framework to control the input pipeline is two folds. First, it improves the training efficiency by a self-adaptive sampling scheme. Second, it guides DON to evolve towards a more effective model automatically.

In contrast to applying RL to solve the problem directly, using RL to make decisions on the sampling probability adjustment is feasible. On one hand, in practice, the action space of this probability adjustment task is smaller than that of directly attacking the whole problem. Since a main property of real-world graph is scale-free, we find that among the vertices there exists skewness regarding the significance to build high-quality solutions so that the pattern of rising up and pushing down vertex sampling probability is not diverse. On the other hand, in DON-RL framework, RL only serves as an auxiliary model to improve the training effectiveness and efficiency of DON, instead of estimating \( \hat{Q}(S, v; \Theta) \) directly. Thus, the effectiveness of DON is less sensitive to low-quality state-action trajectories. Section 5 introduces the RL formulation and our algorithm in details.

### 4 DEEP ORDER NETWORK

In this section, we propose the model Deep Order Network (DON) as a parameterized evaluation function \( \hat{Q}(S, v; \Theta) \) to replace the evaluation function \( Q(S, v) \) in Algorithm 1. Concretely, \( \hat{Q}(S, v; \Theta) \) takes a vectorized representation of partial solution \( S \) of \( w - 1 \) vertices as input, and outputs a vector \( v \in \mathbb{R}^N \) where \( v_i \) represents the likelihood of inserting a vertex \( v_i \) to the permutation, given the current partial solution \( S \). And \( \hat{Q}(S, v; \Theta) \) being learned will replace \( Q(S, v) \) seamlessly in Algorithm 1 to generate a solution.

![Figure 3: The overview of DON-RL. DON-RL contains two components. One is in the blue box named DON. The other is in the orange box named the Policy Network. DON-RL learns a \( \hat{Q}(S, v; \Theta) \) by sampling partial solutions over a graph. The Policy Network takes the sampling probability \( prob_t \) as input to predict a sampling tuning action \( a_t \). The training of the two networks is performed interactively, where the updating of the Policy Network is controlled by the evaluation result of DON.](image-url)
To learn \(\hat{\Omega}(S, v; \Theta)\), we learn a vertex representation with the preservation of quality of the partial solutions, i.e., partial permutations \(\Phi(S)\) which are composed by any subsets of vertices of size \(w\). For a given graph, this representation encodes the hidden optimality structure of vertices in \(F(\Phi(S))\) within a window. Intuitively, vertices with a higher \(\hat{\Omega}\) value tend to cluster in the vector space, whereas vertices with a lower \(\hat{\Omega}\) value are kept away. A high-quality feasible solution can be constructed by set expansion towards a high \(\hat{\Omega}\) value in the vector space. It is worth mentioning that, for a partial permutation \(S\) within a window, the locality score function \(F(\Phi(S))\) should permutation invariant with respect to \(\Phi(S)\). Therefore, the learned evaluation function \(\hat{\Omega}\) is permutation invariant with any permutation of the elements in \(S\).

**Property 1.** [44] A function of sets \(f : X \rightarrow y\), is permutation invariant, i.e., for any set \(x = \{x_1, \cdots, x_M\} \in X\) with any permutation \(\pi\) of the elements in \(x\), \(f(\{x_1, \cdots, x_M\}) = f(\{x_{\pi(1)}, \cdots, x_{\pi(M)}\})\).

DeepSets [44] is a neural network architecture to model the permutation invariant function defined on sets. For a set \(x\) whose domain is the power set of a countable set, any valid, i.e., permutation invariant function can be represented in the form of Eq. (4), for two suitable functions \(\phi\) and \(\rho\).

\[
\begin{align*}
f(x) &= \rho(\sum_{i=1}^{M} \phi(x_i)) \quad (4)
\end{align*}
\]

In Eq. (4), \(x_i \in \mathbb{R}^D\) is the \(i\)-th element in the set \(x\), \(\phi : \mathbb{R}^D \rightarrow \mathbb{R}^K\), and \(\rho : \mathbb{R}^K \rightarrow \mathbb{R}\). Intuitively, \(\phi\) generates the element-wise representation for each \(x_i\), and \(\Sigma\) is a pooling function which performs a generalization of classic aggregation functions, e.g., SUM, AVG and MAX on the set. The added-up representation is processed by \(\rho\) in the same manner as any neural network. In our problem, the domain of partial solutions \(S\) is the combinations of \(V\) with size \(w - 1\), where \(V = \{v_1, v_2, \cdots, v_N\}\) is the vertex set. The evaluation function \(\hat{\Omega}(S, v; \Theta)\) for any such \(S\) can be learned by parameterizing the two functions \((\phi\) and \(\rho))\) with a specified pooling function \(\Sigma\).

We measure the output of \(\hat{\Omega}(S, v; \Theta)\) by a distribution \(p(v|S)\), where \(p(v_i|S)\) is the \(i\)-th element of \(p(v|S)\), meaning how probable the \(i\)-th vertex can be added into the current partial solution \(S\). With the Bayes rule, this probability is in Eq. (5).

\[
\begin{align*}
p(v_i|S) &= \frac{p(v_i,S)}{p(S)} = p(S \cup \{v_i\}) \quad (5)
\end{align*}
\]

In Eq. (5), \(p(S \cup \{v_i\})\) is the marginal probability of the extended solution, which can be estimated by computing \(F\) score of the partial permutation \(\hat{\phi}(S \cup \{v_i\})\) explicitly.

\[
\begin{align*}
p(S \cup \{v_i\}) &= \frac{F(\hat{\phi}(S \cup \{v_i\}))}{\sum_{i=1}^{N} F(\hat{\phi}(S \cup \{v_i\}))} \quad (6)
\end{align*}
\]

Additionally, we use the normalized \(F(\hat{\phi}(S \cup \{v_i\}))\) values (Eq. (6)) to make training more stable. As shown in Fig. 3, a training instance is constructed by sampling a partial solution \(S\) with \(w - 1\) vertices as input feature, then \(p(v|S)\) (Eq. (6)) serves as the soft label for training. We use the cross entropy as the training loss, as given in Eq. (7). Here, \(\hat{p}(v|S)\) is the prediction of DON.

\[
\begin{align*}
L(S) = \sum_{i=1}^{N} -p(v_i|S) \log(\hat{p}(v_i|S)) \quad (7)
\end{align*}
\]
Actions: an action is a vector of 0-1, $a = \{a(i)\}_{i=1}^{n} \in \{0,1\}^{n}$. Each element $a(i)$ denotes a tuning action imposed on $prob(v_{i})$, i.e., increase (0) or decrease (1). There is a tuning rate $\lambda \in \mathbb{R}$ to control the constant delta changes of each element $prob(v_{i})$ for a given $a(i)$. The adjustment is specified in Eq. (8). The final step of an adjustment is the $L1$ normalization of $prob$.

$$prob(v_{i}) = \begin{cases} prob(v_{i}) + \lambda, & a(i) = 0 \\ prob(v_{i}) - \lambda, & a(i) = 1 \end{cases} \quad (8)$$

Transition: given the state-action pair $(prob_{i}, a_{i})$ at time $t$, the transition to next state $prob_{t+1}$ is deterministic.

Reward: the reward $r_{t} = r(prob_{i}, a_{i})$ reflects the benefits of updating a state $prob_{i}$ by an action $a_{i}$ at time $t$. We use the evaluation results (e.g., root mean square error, cross entropy, etc.) of DON trained on the training data generated by updating $prob_{i}$ as $r_{t}$. To avoid the over-fitting of DON, instead of sampling the evaluation set $D_{E}$ according to $prob_{i}$ directly, we adopt the fixed degree sampler, i.e., the initial prob to sample the first vertex and generate the optimal partial solution with window size $w$. The evaluation is performed on a selected evaluation set, $D_{E}$, after training DON a certain number of steps. The cumulative reward is computed by adding future reward discounted by a discounting factor $\gamma \in [0,1]$. Policy: A policy function $\pi(a_{i}|prob)$ specifies a tuning operation $a_{i}$ to perform on a given state $prob_{i}$. Here we employ a parametric function $\pi_{\Theta}(a_{i}|prob)$ with parameter $\Theta'$ to analog output of the policy. This model can be any model any multi-label classification model [38], e.g., multilayer perceptron (MLP), which takes prob as input and outputs the probability of an action $a$. We use $\pi_{\Theta}(a_{i}|s_{i})$ to denote the output probability at time $t$. To distinguish from DON $\hat{U}(S,v,\Theta)$, we use $\Pi(s;\Theta')$ to denote the Policy Network.

5.1 Train Tuning Policy
To find the optimal policy, there are two kinds of RL algorithms [34]. One is value-based, e.g., Q-learning, which learns the optimal state-action values in Eq. (3) and the corresponding optimal policy is to take the best action in any state. The other is policy-based, which learns the optimal policy directly from a collection of policies. Since the policy-based RL algorithm is more suitable for high dimensional action space and has better convergence properties, we use a model-free, policy-based RL algorithm to train the Policy Network directly. The objective is to find the optimal policy $\pi^{*}$ by maximizing the following expected accumulated and discounted rewards of Eq (9), which is estimated by sampling state-action trajectory $s(a_{0}, a_{0}, r_{0}, s_{1}, a_{1}, ..., s_{T})$ of $T$ steps.

$$J(\Theta') = \mathbb{E}[\sum_{t=0}^{T} \gamma^{t} r_{t}|\pi_{\Theta}'] = \mathbb{E}_{s \sim \pi_{\Theta}(s)}[\sum_{t=0}^{T} \gamma^{t} r_{t}] \quad (9)$$

The gradient of the Eq. (9) is formulated using the REINFORCE algorithm [43] as given in Eq. (10).

$$\nabla J(\Theta') = \mathbb{E}_{s \sim \pi_{\Theta}(s)}[\nabla \log \pi(s_{t}) \sum_{t=0}^{T} \gamma^{t} r_{t} - b(s_{t})] \quad (10)$$

Here, as the reward $r$ is sampled from $\pi_{\Theta}(s_{t})$, $\sum_{t=0}^{T} \gamma^{t} r_{t}$ is always positive, a baseline function $b(s)$ is introduced to reduce the variance of the gradients. The formula $\sum_{t=0}^{T} \gamma^{t} r_{t} - b(s)$ indicates whether a reward is better or worse than the expected value we should get from state $s$. For the baseline function $b(s)$, we adopt the widely-used moving average of the historical rewards, i.e., the evaluation result of DON.

The training algorithm of the Policy Network is given in Algorithm 2. It takes a graph $G$, an initial and experience-based sampling probability $prob_{0}$, the evaluation set $D_{E}$, and the learning rate of RL as input, and trains Policy Network $\Pi(s;\Theta')$ by interacting with the training process of Deep Order Network $\hat{U}(S,v,\Theta)$. The intuition is if the reward of an action is positive, it indicates the action is good and the gradients should be applied to make the action even more likely to be chosen in the future. However, if the reward is negative, it indicates the action is bad and the opposite gradients should be applied to make this action less likely in the future.

We briefly explain the algorithm below. First, Policy Network $\Pi(s;\Theta')$ is initialized (line 2). The training of policy gradient starts at line 3. Before that, the training of DON has started several steps to collect enough evaluation results for computing the baseline. In each RL step, the algorithm updates $\Pi(s;\Theta')$ once by one Monte Carlo sampling of a trajectory of length $T$ (line 4). For each time $t$, it repeats the following 4 steps one by one. First, draw a random action $a_{t}$ based on the probability output by Policy Network, i.e., $\pi_{\Theta}(a_{i}|prob_{t})$ (line 5). Instead of directly choosing the action with the highest probability, the random action manages a balance between exploring new actions and exploiting the actions which are learned to work well. Second, apply the action $a_{t}$ on the state $prob_{t}$ by imposing an increment/decrement of $\lambda$ (line 6), to generate the next state $prob_{t+1}$. Third, generate the training data set $D_{t+1}$ of DON by sampling $G$ with probability $prob_{t+1}$, and feed $D_{t+1}$ to train DON (line 7). Fourth, after training $\hat{U}(S,v,\Theta)$ in a certain number of steps, we use the evaluation set $D_{E}$ to evaluate DON, and collect the evaluation result as reward $r_{t}$ (line 8). When a trajectory is simulated, we compute the cumulative rewards (line 11) and apply gradient ascent to update the Policy Network $\Pi(s;\Theta')$ (line 12).
6 EXPERIMENTAL STUDIES

In this section, we present our experimental evaluations. First, we give the specific setting of the testing including datasets, settings of the models and training. Then, we compare the proposed model: DON-RL with the state-of-the-art algorithmic heuristic, conduct an A/B testing to validate the effect of Policy Network, and observe the performance of models as \( w \) varies. Finally, two case studies of compressing graph and deriving a competitive edge partitioning from model-based ordering are presented.

6.1 Experimental Setup

Datasets: We use six real graphs collected from Stanford SNAP [27] and KONECT (The Koblenz Network Collection) [1]: Wiki-vote is the Wikipedia admission vote network. Facebook is the Facebook friendship network. p2p is a snapshot of the Gnutella peer-to-peer file sharing network. Arxiv-HEP is the Arxiv collaboration network between authors in the field of High Energy Physics. Cora is the cora scientific paper citation network. PPI is a protein-protein interaction network where vertices represent human proteins and edges represents physical interactions between proteins.

In addition, to validate the performance of our models on graphs with different structures explicitly, we generate six synthetic graphs by SNAP random graph generator: power-law and Erdős-Rényi. For power-law, we choose three power-law distributions with shape parameter \( \gamma \in \{1.6, 1.8, 2.0\} \) and adopt Newman’s method [30] to generate three power-law graphs: PL10K_1.6, PL10K_1.8 and PL10K_2.0. For Erdős-Rényi, we utilize the parameter \( p \in \{0.02, 0.05, 0.1\} \) which indicates the probability of edge existence between two vertices to generate three synthetic graphs with different densities, that are, ER10K_0.02, ER10K_0.05 and ER10K_0.1. All synthetic graphs have 10,000 vertices. Table 3 summarizes the information of these real and synthetic graphs.

Reduce the Sampling Classes: Recall that DON takes a vertices set sampled from \( V(G) \) as input to predict the likelihood of appending each vertex to the permutation. Based on the RL-based sampling probability tuning approach we have explored, we can further reduce the sampling classes \( V(G) \) by a simple but effective strategy. The reduction is based on the automorphism of vertices on the triad of the graph. We use an example in Fig. 5 to illustrate this reduction. In Fig. 5(a), vertices \( v_1, v_2 \) and \( v_3 \) have a common in-neighbor \( u \), which is also their only neighbor. For \( v_i \) (\( i = 1, 2, 3 \)), we have \( S(v_1, u) = 1 \) and the \( S \) value is 0 for all the other vertices. \( v_1, v_2 \) and \( v_3 \) are automorphism so that permuting them can be regarded as permuting one vertex \( v^* \) three times, where \( S(v^*, u) = 1 \) and \( S \) is 0 for the other vertices. In other words, if three positions for the three vertices \( v_i \) are targeted in the permutation, the \( F \) value will be equivalent for all the 3! assignments of \( v_i \). In this vein, we merge \( v_1, v_2 \) and \( v_3 \) to one vertex \( v^* \) as shown in Fig. 5(b). Sampling vertex sets and inference to generate permutation can be directly conducted on the merged graph (Fig. 5(b)). Each time the model selects \( v^* \) to extend the permutation, we insert a vertex randomly drawn from \( v_1, v_2, v_3 \). For real-world graph, due to its power-low degree distribution, our experiments show this preprocessing can reduce about 9% vertices on average.

Model settings: In DON, both the \( \phi \) and \( \rho \) networks are two-layer MLP. We use the SUM as the pooling layer to add up the representations of \( \phi \) and the output is further processed by \( \rho \). The input of \( \phi \) is an identity vector representation, \( \{0, 1\}^n \), of a partial solution with size \( w = 1 \) and the output is a probability distribution \( \rho(S) \in \{0, 1\}^n \) generated by a softmax function. All the hidden units are activated by ReLU function \( ReLU(x) = \max(0, x) \).

The Policy Network is a multi-label classifier of two-layers MLP. The input is the state \( \text{prob} \in \{0, 1\}^n \) and the output layer predicts the probability to perform action \( a_i \), i.e., a vector \( \{0, 1\}^n \) by sigmoid activation. Similar to DON, the hidden units are activated by ReLU function.

Implementation and training: The learning framework is built on Tensorflow [2] 1.8 with Python 2.7. We use Adam [24] and RMSProp optimizer to train DON and the Policy Network, respectively, which are trained interactively as shown in Algorithm 2.

Table 4 shows the hyper-parameters settings in the training. The models are learned with these parameters tuned in the corresponding experienced range. In Table 4, the global steps and RL steps are the numbers of parameter updating of DON and the Policy Network, respectively. For one RL step, a trajectory of length \( T \) DON training is conducted. To this end, (global steps) / (RL steps)

| Table 3: Datasets |
|-------------------|
| **Graphs** | **|V|** | **|E|** | **Density** | **Description** |
| Wiki-vote [27] | 7,115 | 103,689 | 0.0020 | vote graph |
| Facebook [27] | 4,039 | 88,234 | 0.0108 | social graph |
| p2p [27] | 6,301 | 20,777 | 0.0005 | file sharing graph |
| Arxiv-HEP [27] | 9,877 | 25,998 | 0.0003 | co-authorship graph |
| Cora [1] | 23,166 | 91,500 | 0.0001 | citation graph |
| PPI [27] | 21,557 | 342,353 | 0.0015 | biological graph |
| PL10K_1.6 | 10,000 | 1,121,972 | 0.0024 | power-law graph |
| PL10K_1.8 | 10,000 | 58,934 | 0.0011 | power-law graph |
| PL10K_2.0 | 10,000 | 31,894 | 0.0006 | power-law graph |
| ER10K_0.02 | 10,000 | 1,080,133 | 0.0200 | ER graph |
| ER10K_0.05 | 10,000 | 2,498,836 | 0.0500 | ER graph |
| ER10K_0.1 | 10,000 | 5,004,331 | 0.1000 | ER graph |

| Table 4: The Hyper-parameter Configuration |
|------------------------------------------|
| **Hyper-parameters** | **Values** |
| DON | learning rate \( 10^{-3} - 10^{-4} \) |
| | mini-batch size \( 64 - 256 \) |
| | # hidden units \( 32, 64, 128, 256 \) |
| | global steps \( 5 \times 10^2 - 5 \times 10^3 \) |
| Policy Network | learning rate \( 10^{-3} - 10^{-4} \) |
| | trajectory length \( T \in \{10, 20\} \) |
| | RL steps \( 50 - 300 \) |
| | discounting factor \( \gamma \in \{0.9, 0.95\} \) |
| | tuning rate \( A \in \{0.1\} \) |
| | # hidden units \( 32, 64, 128, 256 \) |
| | # evaluation set \( 2000, 5000 \) |
Table 5: Result of Graph Ordering

| $F(\Phi)$ | $|V'|$ | GO | DON | DON-RL |
|----------|--------|----|------|--------|
| Wiki-vote | 5,880  | 145,736 | 156,871 | 157,669 |
| Facebook  | 3,974  | 230,031 | 207,511 | 234,151 |
| p2p      | 5,624  | 20,472  | 21,422  | 22,086  |
| Arxiv-HEP | 9,877  | 85,958  | 90,629  | 91,001  |
| Cora     | 22,317 | 98,334  | 101,063 | 100,966 |
| PPI      | 19,041 | 383,343 | 347,237 | 404,728 |
| PL10K_1.6 | 8,882  | 166,540 | 190,021 | 197,622 |
| PL10K_1.8 | 8,292  | 66,272  | 86,840  | 89,930  |
| PL10K_2.0 | 8,484  | 29,373  | 37,332  | 38,071  |
| ER10K_0.02 | 10,000 | 136,923 | 145,084 | 162,462 |
| ER10K_0.05 | 10,000 | 615,706 | 673,357 | 724,743 |
| ER10K_0.1  | 10,000 | 2,319,250 | 2,534,032 | 2,647,686 |

6.3 Adaptive Training by RL

In order to validate the effectiveness of the Policy Network, we perform A/B test to observe the effect of RL-based adaptive training, i.e., using the same hyper-parameters to train a DON model and a conjuncted DON-RL model. During the overall training process, we generate a solution after per 10% global training steps of DON ends. We directly observe the changes of objective function $F(\Phi)$ as the loss function of the model is not a direct reflection of the objection of this NP-hard problem. Fig. 6 presents the results over the 12 graphs in Table 3.

We observe that the performance of DON and DON-RL are different among different graphs during training. In general, imposing RL on the sampling probability adjustment improves the performance of DON in most cases. As shown in Fig. 6(a), 6(c), 6(e)-6(g) and 6(i)-6(l), DON-RL can achieve better results than DON in graphs with high density since the locality of vertices is significantly skew on the graph. On the other hand, for the graph with low density and flat degree distribution, such as Cora (Fig. 6(d)) and p2p (Fig. 6(b)), using RL does not make significant improvement during the training. We conjecture, in this scenario, the potential action space is relatively large so that it is difficult for RL algorithm to find an action trajectory with high reward, especially when the locality has been well captured by DON. In addition, due to the dynamic adjusting mechanism, DON-RL can make the training process robust and avoid over-fitting. As shown in Fig. 6(b) and 6(i), DON reaches its best effect much earlier than DON-RL then it goes worse as over-fitting. In contrast, DON-RL can rectify the over-fitting autonomously in Fig. 6(c) and 6(h). The performance of DON-RL increases constantly and surpasses DON in the end. Therefore, the RL-based DON-RL is suitable for the graphs whose structures are highly skew. The results in Fig. 6 also confirms the effectiveness of the Policy Network in DON-RL.

6.4 Varying the Window Size $w$

In this section, we investigate the effect of the window size $w$ in Eq. (1). Fig. 7 shows for graph PL10K_1.6, the $F$ scores of 5 models training with $w$ is set to $\{4, 5, 6, 7, 8\}$. For these DON models, they perform stably better than GO as $w$ grows. In Fig. 7, DON-RL is the 5 models associated with RL training, which generates permutations of highest $F$. Recall that the approximate ratio of GO is $\frac{1}{\sqrt{\text{w}}}$, which means theoretically, the larger the $w$, the larger the gap between GO and the optimal solution, i.e., the improvement space of GO. Interestingly, in Fig 7, as $w$ grows, the increment of locality score of model-based approaches grow a little faster than that of GO. It implies that the model-based approaches can better take advantage of the gap to optimized the learned solution.

In addition, another interesting observation is that models trained with smaller $w$ have better performance than that of larger $w$. It indicates that we can use pre-trained models with small $w$ to generate different permutations of larger $w$.

6.5 Case Studies

In this section, we perform two case studies to demonstrate the power of DON-RL in solving real-world problems.

**Graph Compression:** First, we explore the potential of using DON-RL to deal with graph compression problem. Concretely, given a
where \( \lceil \frac{A}{b} \rceil \) is the number of blocks in \( A \). A good ordering for the compression should have low \( \text{cost}_{nz}(A,b) \) for given block size \( b \).

Fig. 8 demonstrates the compression cost \( \text{cost}_r(A,b) \) on the 5 real graphs in Table 3, when the block wide \( b \) is 8 (Fig. 8(a)), 16 (Fig. 8(b)) and 32 (Fig. 8(c)). We compare the compression performance with two methods: Degree permutes the vertex based on the decreasing degree. SlashBurn [22] is the graph ordering algorithm proposed for graph compression. We set \( w = 5 \) for our method: DON-RL for all datasets. As shown in Fig. 8, DON-RL outperforms the other methods on three datasets (Wikivote, Facebook and Cora) and achieves comparable results on two datasets (p2p and PPI). For the number of nonempty blocks, DON-RL reduces the counts by 2.97× and 2.25× compared with the second best orderings on Facebook and Cora. Interestingly, Degree is a quite good heuristic ordering for the graph compression. It achieves the best performance on p2p and PPI. The results validate that our learned vertex order is a good potential criteria for graph compression.

**Graph Edge Partitioning:** The second application we investigate is graph edge partitioning, which can achieve better load balancing than traditional vertex partitioning for skew graph data in the parallel/distribution environment. The problem is, given a partition number \( k \), partitioning the edge set \( E \) disjointly into \( k \) balanced subsets \( E_i \), to minimize the replication factor (RF) of the vertices in \( k \) partitions:

\[
\text{RF}(E_1, \cdots, E_k) = \frac{1}{|V|} \sum_{i \in [k]} |V(E_i)|, \quad \text{where } \bigcup_{i \in [k]} E_i = E. 
\]

The problem is also NP-hardness [45]. Since the graph ordering is based on vertex locality, we use the ordering result of DON-RL to generate an edge partitioning directly. The partition is conducted.
by assigning edges of adjacent vertices in the permutation to one partition under the balance constraint.

Fig 9 shows the partitioning derives from DON-RL with w = 5, and 5 other edge partitioning and graph ordering algorithms over Facebook. Here, Random [9] assigns each edge independently and uniformly to k partitions. The Greedy [9] heuristic preferentially assigns an edge to a partition already contains the two or one of its end vertices with trading off the balance. NE and SNE [45] are the state-of-the-art edge partitioning heuristic based on neighbor expansion. The replication factor of DON-RL is smaller than that of the widely used Greedy up to 37%. Also, as the number of partition increases, the replication factor of DON-RL increases much slower than Greedy and SlashBurn. It is interesting that a good partitioning derives from a high-quality ordering result.

7 RELATED WORKS

Graph Representation Learning is to learn an effective graph representation to encode sufficient features for downstream machine learning/deep learning tasks. Recently, graph representation learning have been extensively studied. The surveys can be found in [12, 20]. The traditional graph representation learning, that uses matrix factorization to find a low-rank space for the adjacency matrix [3, 7, 15], cannot be used to solve our problem, because they are designed to reconstruct the original graph. Structure preserving representation learning techniques can encode/decode graph structures and properties. DeepWalk [31] and Node2Vec [19] encode the neighborhood relationship by exploiting truncated random walks as embedding context. Furthermore, there are the graph representations to preserve vertex information at different granularity from microscopic neighborhood structure to macroscopic community structure. [35, 37, 46] carry high-order proximity and closeness measures, to provide flexible and diverse representations for unsupervised graph learning tasks. M-NMF [41] incorporates graph macroscopic proximity, the community affiliation of vertex. Such approaches are designed to machine learning tasks (e.g., node classification, link prediction, as well as recommendation). Unfortunately, these representations cannot be directly applied to our problem. First, these representations are learned by unsupervised fashion. Second, they do not contain abundant vertex features, the learned graph proximity is insufficient. Third, our problem aims to find an optimal combinatorial structure instead of performing inference on individual instances which only involves local information.

There are reported studies on representation learning over special graphs, such as representation of heterogeneous graphs [13], dynamic graphs [36], attributed graphs [25]. And there are specified graph machine learning applications that require specialized graph representation, for instance, influence diffusion prediction [14], anomaly detection [21], and network alignment [29]. These models are too specific to deal with our problem.

Neural Networks for Combinatorial Optimization: Early works that use neural network to construct solutions for NP-hard optimization problems are summarized in [33]. Recently, deep learning has also been adopted to solve combinatorial optimization. A new attention-based sequence-to-sequence neural architecture, Pointer Network [40], is proposed and is used to solve the planar Travel Salesman Problem (TSP). Pointer Network learns the planar convex hull supervised by a set of problem instances and solutions alone, and then [8] uses reinforcement learning, the Actor-Critic algorithm [26], to train Pointer Network. Instead of using given training instances, an actor network makes trials of tour and evaluates the trials using a critic network. To further improve the performance, [4] proposes an alternative neural network which encodes input nodes with multi-head attention [39]. These approaches [4, 8, 40] concentrate on 2D Euclidean space TSP, and it is non-trivial to extend them to deal with graphs. For graph data, Dai et. al. [23] propose a framework to learn a greedy meta-algorithm over graphs. First, a graph is encoded by a graph embedding network. Second, the embedding is fed into some more layers to estimate an evaluation function. The overall network is trained by deep Q-learning [43] in an end-to-end fashion. The framework is demonstrated to solve Minimum Vertex Cover, Maximum Cut and TSP. These studies aim to generalize the process of problem-solving for a distribution of problem instances offline. A recent study provides a supervised
learning approach to deal with NP problems equivalent to maximum independent set (MIS). It adopts graph convolutional network (GCN) [25] to predict the likelihood whether each vertex is in the optimal solution [28]. Since it relies on the state-of-the-art MIS local search heuristics to refine the candidate solutions, the bare utility of the model needs to be further studied.

8 CONCLUSION
In this paper, we focus on an NP-hard problem, graph ordering, in a novel, machine learning-based perspective. Distinguished from recent research, the NP-hard problem is over an specific larger graph. We design a new model: Deep Order Network (DON) to learn the underlying combinatorial closeness over the vertices of graph, by sampling small vertex sets as locally partial solutions. To further improve the sampling effectiveness, we propose an adaptive training approach based on reinforcement learning, which automatically adjusts the sampling probabilities. Compare with the high-quality greedy algorithm, our overall model, DON-RL improves the quality of solution up to 7.6% and 35% for real graphs and synthetic graphs, respectively. Our study reveals that a simple neural network has the utility of the model needs to be further studied.